

Mark Levene  
Alexandra Poulovassilis (Eds.)

# Web Dynamics

Adapting to Change in Content, Size,  
Topology and Use



Springer

## Web Dynamics



**Springer-Verlag Berlin Heidelberg GmbH**

Mark Levene  
Alexandra Poulouvassilis

# Web Dynamics

Adapting to Change in Content,  
Size, Topology and Use

With 76 Figures and 29 Tables



Springer

Mark Levene  
Alexandra Poulouvassilis  
School of Computer Science and Information Systems  
Birkbeck University of London; Malet Street  
London WC1E 7HX  
United Kingdom

Library of Congress Cataloging-in-Publication Data applied for  
Die Deutsche Bibliothek - CIP-Einheitsaufnahme  
Bibliographic information published by Die Deutsche Bibliothek  
Die Deutsche Bibliothek lists this publication in the Deutsche  
Nationalbibliografie; detailed bibliographic data is available in the  
Internet at <<http://dnb.ddb.de>>.

**ACM Subject Classification (1998): H.3.3 H.3.5 H.5.4**

**ISBN 978-3-642-07377-9 ISBN 978-3-662-10874-1 (eBook)**  
**DOI 10.1007/978-3-662-10874-1**

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag Berlin Heidelberg GmbH. Violations are liable for prosecution under the German Copyright Law.  
[springeronline.com](http://springeronline.com)

© Springer-Verlag Berlin Heidelberg 2004  
Originally published by Springer-Verlag Berlin Heidelberg New York in 2004  
Softcover reprint of the hardcover 1st edition 2004

The use of designations, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover Design: KunkelLopka, Heidelberg  
Typesetting: Computer to film by author's data

Printed on acid-free paper 45/3142PS 5 4 3 2 1 0

---

## Preface

The World Wide Web has become a ubiquitous global tool, used for finding information, communicating ideas, carrying out distributed computation and conducting business, learning and science.

The Web is highly dynamic in both the content and quantity of the information that it encompasses. In order to fully exploit its enormous potential as a global repository of information, we need to understand how its size, topology and content are evolving. This then allows the development of new techniques for locating and retrieving information that are better able to adapt and scale to its change and growth.

The Web's users are highly diverse and can access the Web from a variety of devices and interfaces, at different places and times, and for varying purposes. We thus also need techniques for personalising the presentation and content of Web-based information depending on how it is being accessed and on the specific user's requirements.

As well as being accessed by human users, the Web is also accessed by applications. New applications in areas such as e-business, sensor networks, and mobile and ubiquitous computing need to be able to detect and react quickly to events and changes in Web-based information. Traditional approaches using query-based 'pull' of information to find out if events or changes of interest have occurred may not be able to scale to the quantity and frequency of events and changes being generated, and new 'push'-based techniques are needed.

In January 2001, we organised a workshop on 'Web Dynamics' in London to explore some of these issues (see [www.dcs.bbk.ac.uk/WebDyn](http://www.dcs.bbk.ac.uk/WebDyn)). Following on from that workshop, we co-edited a special issue of the Computer Networks journal on Web Dynamics (vol. 39, no. 3). In May 2002 we organised a second Web dynamics workshop, this time co-located with WWW'2002 in Hawaii (see [www.dcs.bbk.ac.uk/WebDyn2](http://www.dcs.bbk.ac.uk/WebDyn2)).

Our aim with the second workshop was to continue the momentum built up from the first workshop and the special issue, and to identify major new research advances and challenges. The topics discussed fell into four main areas: models of evolution of the Web's structure, locating and retrieving Web information, Web applications, and adaptive hypermedia. Several of the contributors to this workshop have written

chapters for this book, and we gratefully acknowledge the contributions of all the chapters' authors.

To our knowledge, this is the first book that has an explicit focus on the dynamics of the Web, and our intended audience is all those who have a professional or personal interest in this. For practitioners, the book gives a critical discussion of the current state of the art as well as the most recent research advances, and our aim is that it will be a valuable reference in evaluating current and emerging technologies. For researchers and developers, we hope the book will motivate new research, prototypes and products. For teachers and students, our aim is that the book will be used to support teaching of graduate-level courses on Web technologies, and also by PhD students as a comprehensive reference resource.

Each chapter covers one particular aspect of Web dynamics, first giving a critical review of the state of the art in that area, then taking a more detailed look at one or two particular solutions and concluding with a discussion of some of the major open problems that still need to be addressed.

Chapter 1 gives an introduction and overview that sets the scene for the rest of the book. The chapters that follow it are divided into four parts, essentially covering the same four topics as emerged from the second Web Dynamics workshop:

- evolution of the Web's structure and content (Chaps. 2–5),
- searching and navigating the Web (Chaps. 6–9),
- handling events and change on the Web (Chaps. 10–14), and
- personalised access to the Web (Chaps. 15–18).

Each part is preceded by a short review written by us, summarising that part and pointing to links, similarities and differences between the techniques discussed in the chapters it contains.

Part I focusses on the structural and statistical properties of the Web and covers techniques for measuring the size of the Web (Chap. 2), identifying 'communities' of related Web pages (Chap. 3), models of evolution of the Web viewed as a graph (Chap. 4), and techniques for measuring the rate of change of Web pages (Chap. 5).

Part II focusses on locating and retrieving Web information and covers Web navigation tools (Chap. 6), Web crawlers (Chap. 7), link and content analysis for ranking Web pages (Chap. 8) and techniques for evaluating Web search engines based on the freshness of their indexes (Chap. 9).

Part III discusses techniques for handling events and change in Web applications, and covers languages for defining event-condition-action rules over XML repositories (Chaps. 10 and 11), embedding calls to Web services within XML documents in order to support peer-to-peer data integration (Chap. 12), detection and notification of changes to Web pages (Chap. 13), and reliable middleware for detecting and reacting to events in heterogeneous distributed environments (Chap. 14).

Part IV discusses personalisation of Web information to how it is being accessed and to the user's specific requirements and preferences. It covers architectures for adaptive hypermedia systems (Chap. 15), adaptive educational hypermedia (Chap. 16), personalisation techniques for the mobile internet (Chap. 17), and techniques for learning and predicting users' browsing patterns (Chap. 18).

Many open research issues remain in these areas, and new challenges are continuously arising. We hope that this book will serve to introduce readers to this exciting field and to the possibilities and potential of the dynamic Web.

*Mark Levene*  
*Alex Poulovassilis*  
October 2003

---

# Contents

## Web Dynamics – Setting the Scene

*Mark Levene, Alexandra Poulouvassilis* ..... 1

---

## Part I Evolution of Web Structure and Content

---

**Introduction** ..... 19

### How Large Is the World Wide Web?

*Adrian Dobra, Stephen E. Fienberg* ..... 23

### Methods for Mining Web Communities:

#### Bibliometric, Spectral, and Flow

*Gary William Flake, Kostas Tsioutsoulis, Leonid Zhukov* ..... 45

### Theory of Random Networks

#### and Their Role in Communications Networks

*José Fernando Mendes* ..... 69

### Web Dynamics, Structure, and Page Quality

*Ricardo Baeza-Yates, Carlos Castillo, Felipe Saint-Jean* ..... 93

---

## Part II Searching and Navigating the Web

---

**Introduction** ..... 113

### Navigating the World Wide Web

*Mark Levene, Richard Wheeldon* ..... 117

### Crawling the Web

*Gautam Pant, Padmini Srinivasan, Filippo Menczer* ..... 153

**Combining Link and Content Information in Web Search**

*Matthew Richardson, Pedro Domingos* . . . . . 179

**Search Engine Ability to Cope With the Changing Web**

*Judit Bar-Ilan* . . . . . 195

---

**Part III Events and Change on the Web**

---

**Introduction** . . . . . 219

**An Event–Condition–Action Language for XML**

*James Bailey, George Papamarkos, Alexandra Poulouvassilis, Peter T. Wood* . . . 223

**Active XQuery**

*Angela Bonifati, Stefano Paraboschi* . . . . . 249

**Active XML: A Data-Centric Perspective on Web Services**

*Serge Abiteboul, Omar Benjelloun, Ioana Manolescu,  
Tova Milo, Roger Weber* . . . . . 275

**WebVigiL: An Approach to Just-In-Time Information Propagation  
in Large Network-Centric Environments**

*Jyoti Jacob, Anoop Sanka, Naveen Pandrangi, Sharma Chakravarthy* . . . . . 301

**DREAM: Distributed Reliable Event-Based Application Management**

*Alejandro Buchmann, Christof Bornhövd, Mariano Cilia, Ludger Fiege,  
Felix Gärtner, Christoph Liebig, Matthias Meixner, Gero Mühl* . . . . . 319

---

**Part IV Personalized Access to the Web**

---

**Introduction** . . . . . 353

**A Survey of Architectures for Adaptive Hypermedia**

*Mario Cannataro, Andrea Pugliese* . . . . . 357

**Adaptive Web-Based Educational Hypermedia**

*Paul De Bra, Lora Aroyo, Alexandra Cristea* . . . . . 387

**MP<sup>3</sup> – Mobile Portals, Profiles and Personalization**

*Barry Smyth, Paul Cotter* . . . . . 411

**Learning Web Request Patterns**

*Brian D. Davison* . . . . . 435

**Index** . . . . . 461



---

# Web Dynamics – Setting the Scene

Mark Levene and Alexandra Poulouvassilis

School of Computer Science and Information Systems  
Birkbeck University of London  
Malet Street, London, WC1E 7HX, U.K.,  
{mark,ap}@dcs.bbk.ac.uk

## 1 Introduction

The World Wide Web is the largest hypertext in existence, with a hyperlinked collection of over four billion accessible Web pages, as of late 2003. It is also probably the fastest growing network of this scale the world has ever known. If we include ‘deep’ Web information, i.e. information inaccessible to search engines, such as information stored in databases that can only be accessed through a specialised interface, then the Web is between 400 to 550 times larger than this estimate for the ‘shallow’ Web (see [12]). The Web is becoming all-pervasive in our lives, and we use it for finding information, communication, computation, leisure, business, learning and science. About 10% of the world’s population now has access to the Web, and its number of users is continuously increasing (see [www.gleach.com/globstats](http://www.gleach.com/globstats)).

The Web is thus dynamic in several ways. First, it is a growing network. However, pages and links are not only being added but are also being removed. The contents of Web pages are also being modified. Service providers such as search engines and portals are fighting a continuous battle to deliver high-quality service in the face of this change and growth. Building mathematical models that can predict accurately how the Web is evolving is important, as it then allows us to develop techniques for locating and retrieving Web information that can adapt and scale up to its change and growth.

Second, Web-based applications in areas such as information monitoring and subscription, order processing, business process integration, auctions and negotiation need to be able to detect and automatically react to events or changes in information content by carrying out some further processing. The traditional approach where information consumers periodically submit retrieval requests to information producers in order to find out if events or changes of interest have occurred may not scale up to the volumes of events being generated in such applications. Instead, new techniques are being employed in which information producers automatically notify consumers when events or changes of interest to them occur.

Third, the Web’s users are highly diverse. They have a wide variety of information needs that may change over time, and they can access the Web from a variety of devices

and interfaces, at different times and from different locations. Thus, new techniques are being developed that allow the presentation and content of Web-based information to be adapted and filtered according to users' individual requirements and preferences, and to the way they are accessing the Web.

These major aspects of the dynamic Web are each addressed in one section of this book:

- evolution of the Web's structure and content (Chaps. 2–5),
- searching and navigating the Web (Chaps. 6–9),
- handling events and change on the Web (Chaps. 10–14), and
- personalised access to the Web (Chaps. 15–18).

We next give an overview of each of these topics, setting the scene for the rest of the book. We conclude the chapter with a look to the future and some of the open problems and new emerging challenges.

## 2 Evolution of Web Structure and Content

One way of measuring the size of the Web is to estimate the number of pages that can be indexed, or 'covered', by a major public search engine. This is often referred to as the 'shallow' Web and excludes information residing in searchable databases, which is referred to as the 'deep' Web.

The question of Web coverage by search engines was addressed by Lawrence and Giles in 1998 [49], with a followup reported a year later [50]. The first question they asked was, what is the size of the accessible Web? In 1998 they estimated the size of the Web to be 320 million pages, and in 1999 their estimate increased to 800 million pages. The technique they used is called capture–recapture, which is well known from the statistical literature to measure wildlife populations. This is the motivation underlying the research reported in Chap. 2 of this book, in which Dobra and Fienberg set out to improve Lawrence and Giles's estimates. Other questions that arise here are how much of the Web do search engines cover, and what is the overlap between search engines. New work now needs to be done to update the now outdated statistics discovered by Lawrence and Giles.

The seminal paper by Broder et al. [19] investigated the network structure of the Web and the way it is evolving. They reported on their findings from a Web crawl of over 200 million Web pages and 1.5 billion links, which is the largest scale study of the Web to date. They discovered a 'bow-tie' structure to the Web, its main components being the 'knot' (27.5% of Web pages) and the left and right 'bows' (21.5% each). The knot (the strongly connected component) consists of pages having a directed path from one to the other, and the left bow (the 'in' component) consists of pages having a directed path to somewhere in the knot, while the right bow (the 'out' component) consists of pages having a directed path from somewhere in the knot. The remaining components of the Web are the 'tendrils' and 'tubes' (21.5%) and the disconnected components (8%). The tendrils consist of pages connecting the 'in' and 'out' components directly. The tubes consist of pages going out of the 'in'

component but not into the knot and of pages going directly into the ‘out’ component. The disconnected components are ones which are unattached to the main bow-tie structure.

Saying that a property such as the in-degree of the Web graph has a *power law distribution* means that the probability of attaining a particular size, say a Web page having an in-degree  $n$ , is proportional to  $1/n^\alpha$ , where  $\alpha$  is greater than or equal to one. Power law distributions are also known as ‘heavy-tailed’ distributions since their tails do not decay exponentially fast as in a normal (Gaussian) distribution. Broder et al. discovered power law distributions for the in-degree of Web pages (exponent 2.1), the out-degree (exponent 2.72) and the strongly connected components of the Web graph (exponent 2.54).<sup>1</sup>

These discoveries have since then resulted in further research into models that explain the emergence of these patterns. Other power law distributions in the Web that have been discovered include: the number of visitors to a site during a day (exponent 2.07) [4], the number of links clicked by Web surfers (exponent 1.5) [43], and the PageRank metric (exponent 2.1) [58].

Broder et al. also computed the average shortest path between pairs of Web pages that are connected by a path of links, and discovered that it was 16.2 when surfers are not allowed to backtrack along links (i.e. treating the Web as a directed graph) and only 6.8 when the direction of links is ignored (i.e. treating the Web as an undirected graph). They thus showed that the Web is a *small-world network*, popularly known through the notion of ‘six degrees of separation’.

Recent progress in this area is discussed by Mendes in Chapter 4, taking a statistical physics perspective. The area of random evolving networks has attracted researchers from many disciplines. Recent research is reported in [6, 32, 55], and [11, 64] are recent popular accounts.

Search engine crawlers navigate the Web continuously, with the aim of keeping search engine indexes as fresh as possible and covering as much as the accessible Web as possible. In order for Google to refresh its search index of about 3 billion Web pages on a monthly basis, its crawler must process about 1,000 Web pages per second. To maintain an index that is as fresh as possible, the crawler must follow a schedule of page revisits that minimises the number of pages that are out of date. Measuring the frequency of change in Web pages is thus a problem that impacts significantly on Web crawlers.

In [17] a stochastic model for monitoring the rate of change of Web pages was devised. If we assume a constant rate of change, then we can model the change with a Poisson process, where the probability of change at the current step is taken to be independent of the time when the last change occurred. This allows us to estimate the expected age of a Web page, whose actual value is zero if the page is up-to-date and is otherwise the elapsed time since the page was last modified [28]. In reality, different groups of pages change at different rates, so to obtain a more accurate model we can

<sup>1</sup> We refer readers who are unfamiliar with the concepts of graph theory to a book such as [65].

assume a constant rate of change only within each group of pages rather than for the Web as a whole.

These and other related issues are dealt with by Baeza-Yates, Castillo and Saint-Jean in Chap. 5, based on a study of the Chilean Web carried out during 2000 and 2001. In Chap. 9 the issue of freshness of a search engine's index provides motivation for Bar Ilan to define new metrics for evaluating the performance of search engines, which take into account the freshness of the Web pages retrieved. Recent research [37] has shown that although most Web pages do not undergo much change, larger Web pages tend to change more often and to a larger extent than smaller ones. Moreover, change is more frequent and extensive for pages whose top-level domain is `.com` or `.net` than for pages from other domains such as `.edu` and `.gov`. This research has also shown that past change to a Web page is a good indicator of future change, which is encouraging for prediction purposes.

A question that researchers have been addressing since the inception of the Web is how to take into account the Web's hyperlinked structure in order to identify high-quality Web pages and communities of Web pages focussed on a common topic or theme.

In his seminal paper, Kleinberg [46] introduced a search algorithm called Hyperlink Induced Topic Search (HITS) that identifies and scores two types of Web page: *hubs*, which are resource lists for a given topic, and *authorities*, which are good sources of information on a given topic. HITS operates on a *focussed subgraph*, which is composed of the set of Web pages returned from a query to a search engine extended by one level of incoming and outgoing links, and with links between pages within the same site removed. It computes the scores of pages through a mutually reinforcing relationship between hubs and authorities such that a good authority is linked to by good hubs and a good hub links to good authorities. HITS is query-dependent and thus it could be incorporated into the workings of a search engine, but only if it can be optimised to run efficiently at query time, i.e. at the time a user submits a query to the search engine.

Other applications of HITS include finding Web communities, by identifying densely focussed subgraphs; finding pages related to a given page, by computing the focussed subgraph starting from a single Web page rather than a set of pages returned by a query; and populating categories in Web directories, by computing the focussed subgraph, for example, using the label of a category as the query, or starting from some pages that are known to be in a given category [25, 41]. In Chap. 3 Flake et al. give an alternative definition of a Web community as a set of Web pages such that each page within the community has more links to other members of the community than links to pages outside the community. They devise an algorithm to construct Web communities that is competitive with HITS.

A query-independent metric for computing a page's authority or importance is the PageRank algorithm introduced by Brin and Page as part of Google's ranking function [18] – we discuss it in more detail in the next section. Dhyani et al. [31] give a recent survey of various metrics that have been designed not only to measure properties of the Web graph but also Web page significance and similarity, and metrics

based on Web-usage patterns. A review of the Web's structure and its similarities to social networks is given by Kumar et al. [48].

### 3 Searching and Navigating the Web

Users can locate information on the Web in essentially three different ways. First, there is *direct navigation*, where users enter the URLs of the Web pages they wish to see directly into the Web browser. In this case the users know exactly what pages they wish to view. Direct navigation also covers the use of bookmarks and the history list, which are the basic navigation aids provided by browsers. Second, there is *surfing* or navigating by following links. In this case the user may start from a home page of a site, reached, for example, through a search engine query or by direct navigation, and then click on links following the cues given by link text. Third, there is the use of a search engine, where the search engine acts as an information gatekeeper that holds the key to finding the Web pages with the information the user wishes to inspect.

According to the Web analytics company WebSideStory, as of March 2003 approximately 65.5% of users locate information by direct navigation, 21% by following links and 13.5% by using search engines. These statistics indicate that many users know where they want to go, for example, when visiting their favourite news site or shopping through their usual e-commerce site. It also shows that navigation is a significant process, since search engines often lead us only to the home page of a site, requiring us to follow some links to find the exact information we are seeking. The statistics also show that there is a substantial growth of e-commerce referrals coming from search engines, which is good news for search engines since their main revenue stream is from advertising.

It is essential to provide tools for search and retrieval of Web information that scale with its growth and cater to a diverse set of user needs. One problem that remains unsolved despite many research efforts is the problem of users getting 'lost in hyperspace' as they are navigating the Web. Issues relating to navigation were looked at by the hypertext community well before the Web era. Nielsen's book on hypertext [57] is an excellent pre-Web introduction to hypertext, which discusses many of the problems we now confront on a day-to-day basis on the Web.

One interesting approach to information seeking, based on the theory of information foraging, is advocated by researchers from the Human Interface research group at Xerox Parc. This cognitive approach is based on the idea that users satisfy their information needs by following the 'scent' of information present in clues such as link text, snippets of text and images [27]. It remains to be seen which approach will ultimately succeed in making headway in resolving the navigation problem. An algorithmic approach based on extending search engine technology to suggest relevant trails that users should follow given an input query is presented by Levene and Wheeldon in Chap. 6.

The PageRank metric discussed in the previous section can be explained via a random surfer model, as a random walk on the Web graph. Under this model PageRank provides a measure that is relevant both for search and navigation. Under the random

surfer model, a surfer clicks on links uniformly at random until he or she gets bored and then jumps at random to another Web page and resumes surfing. The PageRank of a Web page is the proportion of time a surfer will arrive at the page during a navigation session. More formally, the Web, suitably modified to take into account random jumps, can be viewed as a Markov chain and the PageRank vector is its stationary distribution.<sup>2</sup>

According to Kamvar et al. from the PageRank group Stanford University, because the sheer size of the Web, computing the PageRank values can take Google several days. These researchers have found ways to speed-up the computation of PageRank by a factor of two or higher, which is significant given the number of pages with which Google has to deal [44]. This research was considered important enough for Google to recently acquire the technology coming out of this group – see [www.google.com/intl/fi/press/pressrel/kaltix.html](http://www.google.com/intl/fi/press/pressrel/kaltix.html).

In Chap. 8, Richardson and Domingos propose a query-dependent version of PageRank which combats the problem of topic drift, which is the problem of returning Web pages that are peripheral to the topic of the search query. To make their approach scalable they also propose a method to compute their query-dependent PageRank offline for each of the keywords stored in a search engine's index. A related proposal is topic-sensitive PageRank [39], where the PageRank is computed for a selected of topics rather than for all queries.

Search engine crawlers aim to keep search engine indexes as fresh as possible and covering as much of the accessible Web as possible. The strategy a crawler uses in order to prioritise which pages to visit first determines the contents of the index; current wisdom being that using a breadth-first crawling approach is highly competitive [54]. Focussed crawlers are a relatively new breed of crawlers that retrieve Web pages within a specific topic and generally follow a best-first heuristic rather than a breadth-first one. Pant, Srinivasan and Menczer discuss these and other issues relating to Web crawling in Chap. 7. One important issue is the evaluation of different crawling strategies using different measures of page importance, such as the presence of specific keywords, the similarity to a search query, a classifier score or the PageRank score. The standard information retrieval measures of *precision* (fraction of retrieved pages that are relevant) and *recall* (fraction of relevant pages that are retrieved) need to be tailored to the task of Web crawling, taking into account the limited resources of a crawler.

Evaluating the quality of Web search needs to take into account the dynamic nature of the Web. In the context of a Web search engine, the traditional measure of recall relates to the coverage of a search engine, as determined by its crawling quality and capacity, rather than its retrieval algorithms. Thus, because of the sheer size of the Web and the fact that most search engine users inspect only one page of query results, measuring search engine quality is normally done through precision-oriented metrics [40]. In Chap. 9, Bar-Ilan makes a strong argument for new evaluation metrics that measure how a search engine copes with change. Search engine results fluctuate in

<sup>2</sup> A book on stochastic processes such as [62] is useful for readers wishing to gain an understanding of the underlying theory.

time, so for a given search query some new results will appear and some old results will disappear. There is a need to measure the stability of a search engine, in the sense that existing Web pages are not dropped from its index, its index is updated with new information as soon as possible, and duplicate pages are detected. These and related issues are discussed in Chap. 9.

Arasu et al. [9] provide a comprehensive account of Web search engine design, including crawling and indexing. A relatively recent book on information retrieval is [10], which includes amongst other material an evaluation of retrieval performance. Henzinger et al. [42] discuss some challenges for Web search engines such as spam detection, evaluation of the quality of Web pages, duplicate host detection and making the best use of structure when present in Web pages.

## 4 Handling Events and Change on the Web

Originally conceived as a means of exchanging and sharing information, the Web has now become all-pervasive in our lives and supports huge numbers of applications in business, leisure, science and learning. Many of these applications need to be *reactive*, i.e. to be able to detect the occurrence of specific events or changes in information content, and to respond by automatically executing the appropriate application logic. Several examples of such applications are discussed in Part 3 of the book, including information personalisation and classification, information monitoring and subscription, reservations and orders processing, supply chain management, information integration, business process integration, auctions, negotiations, distributed software development, sensor networks and ubiquitous computing.

The traditional approach to determining whether an event or change has occurred would be for an information consumer to submit a retrieval request to the appropriate information producer or producers and to inspect the results returned, perhaps comparing them with earlier results returned from similar retrieval requests. This is information *pull* and has two drawbacks: first, it may not detect the event or change quickly enough, which may have serious consequences in time-critical scenarios; second, it may result in unnecessary requests for information when in fact no event or change of relevance has occurred, resulting in a waste of resources.

Increasingly therefore, information *push* is being employed in Web applications, whereby information producers automatically notify consumers of information in which they have registered an interest, either explicitly or implicitly via user profiles. Jacob et al. discuss the pull and push paradigms in Chap. 13, both in general and specifically in the context of detecting changes to Web pages. They discuss three different ways in which a system can be extended with push capabilities: by direct modification of its source code, by using intelligent agents interfacing with the system's users to extend the system's reactive capabilities, or by providing a wrapper over the system.

*Event–Condition–Action* (ECA) rules are one way of recording consumers' information needs and how to respond to them. Also known as *active rules* or *triggers*,

ECA rules are supported in some form by all the major relational Database Management System (DBMS) vendors. They make a database able to respond automatically to the occurrence of database updates by executing preprogrammed actions provided specified conditions hold. These actions may in turn cause further database updates to occur, which may in turn cause more ECA rules to execute. Thus, ECA rules turn a ‘passive’ database into an ‘active’ database.

ECA rules have been used in conventional database applications for information integration, generating and incrementally maintaining materialised views, enforcing integrity constraints, and maintaining audit trails. ECA rules have also been proposed for use in a variety of Web applications: for specifying and implementing business processes [1, 3, 24, 63], for encoding users’ preferences in information personalisation and information push [24, 60, 5, 13, 14, 36], for encoding negotiation strategies [35] and for supporting change notification [53, 34] and auctions [29].

There are several advantages in using ECA rules to support such functionality, as opposed to implementing it directly in application code. First, ECA rules allow an application’s reactive functionality to be specified and managed within a single rule base rather than being encoded in diverse application programs. This enhances the modularity, maintainability and extensibility of the application. Second, ECA rules have a high-level, declarative syntax and are thus amenable to analysis and optimisation techniques which cannot be applied if the same functionality is expressed directly in application code. Third, ECA rules are a generic mechanism that can abstract a wide variety of reactive behaviours, in contrast to application code that is typically specialised to a particular kind of reactive scenario. Bailey et al. give a review of ECA rules in Chap. 10, discussing how they are used in conventional database applications, their syntax and semantics, and techniques for analysing their run-time behaviour.

However, the use of ECA rules in Web applications poses several new challenges:

- XML is increasingly being used for storing and exchanging information on the Web. There is thus an emerging need for the support of ECA rules over XML repositories. New challenges arise in designing and implementing XML ECA languages because the insertion of a subdocument into an XML document, or the deletion of a subdocument from it, may trigger a set of different events; this is in contrast to, say, a relational database where an INSERT, DELETE or UPDATE action on a table can trigger only the corresponding INSERT, DELETE or UPDATE event. This issue is discussed by Bailey et al. in Chap. 10, and by Bonifati and Paraboschi in Chap. 11. These two chapters present two languages for defining ECA rules on XML repositories, focussing on the syntax of events, conditions and actions, how such rules are executed and prototype implementations.
- One of the key recurring themes relating to the effective deployment of ECA rules in systems is the need for techniques and tools to analyse their run-time behaviour before they are deployed [23, 47]. New challenges arise in developing analysis techniques for XML ECA rules because of the more complex association between updates and event occurrences. Chap. 10 discusses this issue, and the authors present techniques for analysing the run-time behaviour of their XML ECA rules.



- Another recurring theme in the successful deployment of ECA rules is the efficiency of ECA rule execution [23, 47]. ECA rules are a generic technology that subsumes a wide variety of reactive behaviours, and so they are likely to incur a performance penalty compared to special-purpose solutions. The problem becomes more acute in the context of Web applications, where there may be much larger volumes of ECA rules than in traditional database applications and much larger volumes of events being generated.

*Publish/subscribe* systems address this problem by allowing information consumers to register their interest in particular events (via ‘subscriptions’), and information producers to register occurrences of events. Recent research has focussed on encodings of event and condition queries and efficient matching of event occurrences against them [53, 34, 60, 26, 22, 36] and promises to achieve the necessary scalability in the processing of ECA rules in Web applications.

In Chap. 13, Jacob et al. discuss using ECA rules for encoding users’ interests in changes to the content of specific Web pages, and for notifying users of such changes. The chapter focusses particularly on the efficiency of detecting changes to the Web pages, describing a system that supports this functionality and discussing its architecture and implementation.

- Further challenges arise when ECA rules are deployed over distributed as opposed to centralised information repositories, stemming from the distribution, autonomy and heterogeneity of the information sources.

Buchmann et al. explore some of these issues in Chap. 14, and possible solutions. They discuss the dissemination of events in distributed systems, covering the three main approaches: channel-based, subject-based and content-based. Also discussed are the representation and detection of composite events, the management of the events being produced and consumed and transactional issues arising in distributed heterogeneous environments. Chap. 14 then goes on to discuss one particular middleware platform that supports a publish/subscribe event notification service, a reactive functionality service and mechanisms for transaction support and management support.

A major application area for ECA rules is to integrate data from a variety of sources into a single repository. For example, Xyleme [56, 2] is an XML warehouse that allows users to subscribe to changes in Web documents of interest. A set of ‘alerters’ monitor occurrences of simple changes to Web documents. Evaluation of more complex event queries then occurs, followed by notification of these events to an ECA rule execution engine which then performs the necessary actions.

In Chap. 12, Abiteboul et al. present a more general solution to the problem of XML data integration in Web applications, presenting a framework that allows calls to *Web services* to be embedded within XML documents. XML and Web service technologies are being developed and adopted with the aim of providing standard ways for Web-based applications to exchange information and interact with each other (see [www.w3c.org/XML](http://www.w3c.org/XML) and [www.w3.org/2002/ws/](http://www.w3.org/2002/ws/); see also [8] for a recent book on Web services).

In contrast to the traditional approach to data integration, which relies on the formation of ‘global’ schemas in order to integrate information from several sources, the approach described in Chap. 12 undertakes *peer-to-peer* integration of XML data that may be distributed across any number of autonomous nodes in the Web network, each of which is able both to make Web service calls and to accept calls made to its own set of Web services. When such a Web service is called, the XML document in which the call is embedded is expanded with the results returned. Special attributes on elements are used to control the timing of calls, thereby capturing a variety of integration scenarios, including virtual integration, materialised integration and partly materialised, partly virtual.

## 5 Personalised Access to the Web

The Web is accessed by highly heterogeneous, globally distributed users, with different interests and aims, using different devices and interfaces. In order to maximise the usefulness of the information that users retrieve from the Web, this information therefore needs to be filtered and adapted to the requirements of individual users.

Research into *adaptive hypermedia* [20] aims to build models of the users of a hypermedia system and to use these models for the purpose of adapting the system to users’ needs and preferences. Three aspects of a hypermedia system can be adapted: the presentation of information, the information content and the link structure between items of information. Cannataro and Pugliese give a comprehensive description of adaptive hypermedia in Chap. 15, covering some existing architectures and major systems as well as looking in detail at one particular system.

The theme of adaptive hypermedia is continued by DeBra, Aroyo and Cristea in Chap. 16, which focusses more specifically on adaptive hypermedia in the education domain. Here, educational material needs to be tailored to the needs, goals and prior knowledge of individual learners. In addition to personalising the presentation, content and link structure in the educational material, adaptation is also needed to accommodate different learners’ preferred learning styles.

The information domain addressed by the educational hypermedia system is divided into a set of ‘concepts’ that are typically structured into a hierarchy. The user model represents how much an individual learner knows about each concept. As users access fragments of the hypermedia, so their knowledge of concepts improves. In the AHA! system described by DeBra et al., the adaption of user models as users access educational material is specified by means of a set of ECA rules. They also describe a courseware tool that uses *ontologies* as the basis for user models, course structure, and the information domain, thereby facilitating automated reasoning between these different parts of the system as well as reuse of system components between different authors.

Personalisation can also be applied to navigation. *Web usage mining* is concerned with analysing Web log data to find patterns in users’ navigation behaviour in order to then suggest possible navigation trails to users [15, 51, 21, 33]. In Chap. 18, Davison discusses building user models that will allow prediction of the next Web page that

users will request given knowledge of their previous page requests. This kind of prediction can be used for optimisation purposes by prefetching and caching Web pages, as well as for making suggestions to users.

There is growing interest in search engine personalisation, in an attempt to further increase the precision of search results [45]. A further incentive for search engines to provide such a service is the highly competitive nature of the search market, where user loyalty is relatively low. Personalisation comes in two flavours: *explicit*, using information input by the user stating their preferences and giving their demographic details, or *implicit*, using data mining techniques on information collected from Web log data about the user's activities.

*Adaptive Web sites* utilise techniques from both adaptive hypermedia and Web usage mining and aim to automatically improve their structure and presentation through the use of machine learning techniques [61, 7]. One objective is to minimise the cost of locating Web pages, where the expected cost for a page may be measured in terms of the frequency of visits to that page multiplied by the number of clicks necessary to reach the page starting from the Web site's home page.

Reducing the number of clicks is particularly important in mobile portals, where screen size limitations may result in deeply nested menu structures. This topic is discussed in Chap. 17, where Smyth and Cotter build up models of users' navigation preferences in mobile portals in order to predict a user's most likely next options and hence modify the portal's menu structure by promoting these options to higher up the menu hierarchy.

In building user models for the purposes of prediction, it is possible to build one model for each individual user (as in the approaches described in Chaps. 16 and 17), one model representing the typical user (as in the approach described in Chap. 18 and in adaptive Web sites), or a set of stereotypical profiles with respect to which users are classified (as in the approach described Chap. 15). These models are built using machine learning techniques such as Bayesian and neural networks, as well as pattern recognition methods such as clustering and classification [38].

An approach that works for communities of users is *collaborative filtering* [16, 30] which provides recommendations to a user based on the preferences of other similar users. This is now widely used to recommend books and music to consumers [52]. Such systems make use of explicit user ratings on items, in addition to implicit feedback obtained from clickstream data. These inputs are used to train the recommender algorithm prior to making new suggestions to the user.

## 6 The Outlook

The field of Web dynamics is still young, and many of the ideas discussed in this book are still maturing. For a technology that is just over a decade old the achievements to date are impressive, and many of them are driven by commercial needs to deploy online technologies that did not exist prior to the Web. There is a growing demand, not only from the commercial sector but also from government and other institutions, to provide stable and scalable online services.

Portals and search engines provide an entry point to the Web for many users, and Web technologies are constantly being challenged to keep up with the growth and change of the Web in order to provide users with fresh and reliable information. Medium to large Web sites are faced with similar problems, and in order to cater to a global and disparate user base there is a strong need to deploy technologies that deal with change in a seamless and intelligent manner. These challenges have three facets: relating to the Web as a network, the applications that run over the Web, and the Web's users.

Issues relating to studying the evolution of the Web will remain important, for example, a model that unifies all aspects of Web evolution and is consistent with the real Web is yet to emerge. Predicting how the Web is changing at different levels of granularity, from the components within an individual Web page to the Web as a whole, would allow search engine crawlers to prioritise their operation and Web sites and portals to better plan their maintenance policies. As it is not clear if and when the Web's growth will slow down, the ability to detect and promote high-quality Web material will become more important. There is also the issue of dealing with 'spam', already a major problem for the Internet as a whole.

Technologies that support personalisation and collaboration will become more pervasive as the competition to gain the user's attention will not decrease, and users' expectations are growing as the quality of Web technologies is improving. The Web is not only a communication network, it is also a social network, and we expect more technologies to emerge that identify and support different communities of Web users. Mobile technologies such as the ubiquitous mobile phone have widened the reach of the Web and there is a strong need to adapt and integrate applications so that they are operable across the different platforms with which users interact.

New applications and technologies such as sensor networks and ubiquitous computing are resulting in the development of new techniques for handling events and changes in Web-based information, leading to a *reactive Web*. We expect that future research will need to focus on areas such as language design, performance optimisation, security, robustness and interoperability of reactive Web-based applications.

Finally, the *Semantic Web* initiative, with languages such as RDF/RDFS and OWL (see [www.w3.org/2001/sw/](http://www.w3.org/2001/sw/)), is aiming at a more precise and standardised description of Web-based resources, and it promises to lead to better validation, sharing, and personalisation of Web-based information. We expect that future research will also need to focus on the convergence of the reactive and the semantic Web, and some initial work in this area is reported in [59].

In conclusion, as this book demonstrates, the field of Web dynamics is by its very nature highly interdisciplinary, and we believe that it is the collaboration of, and cross-fertilisation between, experts across many fields that will yield the necessary advances.

## References

1. S. Abiteboul, B. Amann, S. Cluet, A. Eyal, L. Mignet, and T. Milo. Active views for electronic commerce. In *Proc. 25th Int. Conf. on Very Large Databases*, pages 138–149, 1999.
2. S. Abiteboul, S. Cluet, G. Ferran, and M.-C. Rousset. The Xyleme project. *Computer Networks*, 39:225–238, 2002.
3. S. Abiteboul, V. Vianu, B.S. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *JCSS*, 61(2):236–269, 2000.
4. L.A. Adamic and B.A. Huberman. Zipf’s law and the internet. *Glottometrics*, 3:143–150, 2002.
5. A. Adi, D. Botzer, O. Etzion, and T. Yatzkar-Haham. Push technology personalization through event correlation. In *Proc 26th Int. Conf. on Very Large Databases*, pages 643–645, 2000.
6. R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.
7. D.J. Aldous. Reorganizing large Web sites. *American Mathematical Monthly*, 108:16–27, 2001.
8. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services Concepts, Architectures and Applications*. Springer, Berlin Heidelberg New York, 2004.
9. A. Arasu, J. Cho, . Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the Web. *ACM Transactions on Internet Technology*, 1:2–43, 2001.
10. R.A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press and Addison-Wesley, Reading, MA, 1999.
11. A.-L. Barabási. *Linked: The New Science of Networks*. Perseus Publishing, Cambridge, MA, 2002.
12. M.K. Bergman. The deep Web: Surfacing hidden value. White paper, Bright Planet, July 2000.
13. A. Bonifati, S. Ceri, and S. Paraboschi. Active rules for XML: A new paradigm for e-services. *VLDB Journal*, 10(1):39–47, 2001.
14. A. Bonifati, S. Ceri, and S. Paraboschi. Pushing reactive services to XML repositories using active rules. In *Proc. 10th World Wide Web Conference*, 2001.
15. J. Borges and M. Levene. Data mining of user navigation patterns. In B. Masand and M. Spiliopoulou, editors, *Web Usage Analysis and User Profiling*, Lecture Notes in Artificial Intelligence (LNAI 1836), pages 92–111. Springer, Berlin Heidelberg New York, 2000.
16. J.S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predicitive algorithms for collaborative filtering. In *Proceedings of Conference on Uncertainty in Artificial Intelligence*, Madison, Wisc., 1998.
17. B.E. Brewington and G. Cybenko. Keeping up with the changing Web. *IEEE Computer*, 33:52–58, 2000.
18. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of International World Wide Web Conference*, pages 107–117, Brisbane, 1998.
19. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, A. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the Web. *Computer Networks*, 33:309–320, 2000.
20. P. Brusilovsky, A. Kobsa, and J. Vassileva, editors. *Adaptive Hypertext and Hypermedia*. Kluwer, Dordrecht, 1998.
21. R.E. Bucklin, J.M. Lattin, A. Ansari, S. Gupta, D. Bell, E. Coupey, J.D.C. Little, C. Mela, and A. Montgomery. Choice and the internet: From clickstream to research stream. *Marketing Letters*, 13:245–258, 2002.

22. A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proc. PODC 2000*, pages 219–227, 2000.
23. S. Ceri, R. Cochrane, and J. Widom. Practical applications of triggers and constraints: Success and lingering issues. In *Proc. 26th Int. Conf. on Very Large Databases*, pages 254–262, 2000.
24. S. Ceri, P. Fraternali, and S. Paraboschi. Data-driven one-to-one Web site generation for data-intensive applications. In *Proc. 25th Int. Conf. on Very Large Databases*, pages 615–626, 1999.
25. S. Chakrabarti, B. Dom, S.R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, D. Gibson, and J.M. Kleinberg. Mining the Web’s link structure. *IEEE Computer*, 32:60–67, 1999.
26. J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCq: a scalable continuous query system for internet databases. In *Proc. 2000 ACM SIGMOD Int. Conf. on Management of Data*, pages 379–390, 2000.
27. E.H. Chi, P. Pirolli, and J.E. Pitkow. The scent of a site: A system for analyzing and predicting information scent, usage, and usability of a Web site. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, pages 161–168, The Hague, Amsterdam, 2000.
28. J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 117–128, Dallas, Tx., 2000.
29. M. Cilia and A.P. Buchmann. An active functionality service for e-business applications. *ACM SIGMOD Record*, 31(1):24–30, 2002.
30. W.W. Cohen and W. Fan. Web-collaborative filtering: Recommending music by crawling the Web. In *Proceedings of International World Wide Web Conference*, Amsterdam, 2000.
31. D. Dhyani, W.K. Ng, and S.S. Bhowmick. A survey of Web metrics. *ACM Computing Surveys*, 34:469–503, 2002.
32. S.N. Dorogovtsev and J.F.F. Mendes. Evolution of networks. *Advances in Physics*, 51:1079–1187, 2002.
33. M. Eirinaki and M. Vazirgiannis. Web mining for Web personalization. *ACM Transactions on Internet Technology*, 3:1–27, 2003.
34. E. Hanson et al. Scalable trigger processing. In *Proc 15th Int. Conf. on Data Engineering (ICDE’99)*, pages 266–275, 1999.
35. J. Hammer et al. The IDEAL approach to internet-based negotiation for e-business. In *Proc 16th Int. Conf. on Data Engineering (ICDE’2000)*, pages 666–667, 2000.
36. F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K.A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe. In *Proc. ACM SIGMOD Conf*, 2001.
37. D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener. A large-scale study of the evolution of Web pages. In *Proceedings of International World Wide Web Conference*, Budapest, 2003.
38. D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. Morgan-Kaufmann, San Francisco, CA, 2001.
39. T.H. Haveliwala. Topic-sensitive PageRank. In *Proceedings of International World Wide Web Conference*, pages 517–526, Hawaii, 2002.
40. D. Hawking, N. Craswell, P. Bailey, and K. Griffiths. Measuring search engine quality. *Information Retrieval*, 4:33–59, 2001.
41. M.R. Henzinger. Hyperlink analysis for the Web. *IEEE Internet Computing*, 1:45–50, 2001.
42. M.R. Henzinger, R. Motwani, and C. Silverstein. Challenges in Web search engines. *SIGIR Forum*, 36, Fall 2002.

43. B.A. Huberman, P.L.T. Pirolli, J.E. Pitkow, and R.M. Lukose. Strong regularities in World Wide Web surfing. *Science*, 280:95–97, 1998.
44. S.D. Kamvar, T.H. Haveliwala, C.D. Manning, and G.H. Golub. Exploiting the block structure of the Web for computing PageRank. Technical report, Department of Computer Science, Stanford University, March 2003.
45. Y. Khopkar, A. Spink, C.L. Giles, P. Shah, and S. Debnath. Search engine personalization: An exploratory study. *First Monday*, 8(7), 2003. [http://www.firstmonday.dk/issues/issue8\\_7/khopkar/](http://www.firstmonday.dk/issues/issue8_7/khopkar/).
46. J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46:604–632, 1999.
47. A. Kotz-Dittrich and E. Simon. Active database systems: Expectations, commercial experience and beyond. In N. Paton, editor, *Active Rules in Database Systems*, pages 367–404. Springer, Berlin Heidelberg New York, 1999.
48. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The Web and social networks. *IEEE Computer*, 35:32–36, 2002.
49. S. Lawrence and C.L. Giles. Searching the World Wide Web. *Science*, 280:98–100, 1998.
50. S. Lawrence and C.L. Giles. Accessibility of information on the Web. *Nature*, 400:107–109, 1999.
51. M. Levene, J. Borges, and G. Loizou. Zipf’s law for Web surfers. *Knowledge and Information Systems*, 3:120–129, 2001.
52. G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7:76–80, 2003.
53. L. Liu, C. Pu, and W. Tang. Supporting internet applications beyond browsing: Trigger processing and change notification. In *Proc 5th Int. Computer Science Conf (ICSC’99), Hong Kong*, pages 294–304, 1999.
54. M. Najork and J.L. Wiener. Breadth-first crawling yields high-quality pages. In *Proceedings of International World Wide Web Conference*, pages 114–118, Hong Kong, 2001.
55. M.E.J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.
56. B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda. Monitoring XML data on the Web. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 437–448, 2001.
57. J. Nielsen. *Hypertext and Hypermedia*. Academic Press, Boston, MA, 1990.
58. G. Pandurangan, P. Raghavan, and E. Upfal. Using PageRank to characterize Web structure. In *Computing and Combinatorics Annual International Conference*, pages 330–339, Singapore, 2002.
59. G. Papamarkos, A. Poulouvassilis, and P.T. Wood. Event-condition-action rule languages for the semantic Web. In *Proc. Workshop on Semantic Web and Databases, at VLDB’03*, 2003.
60. J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient matching for Web-based publish/subscribe systems. In *Proc 7th Int. Conf. on Cooperative Information Systems (CoopIS’2000)*, pages 162–173, 2000.
61. M. Perkowitz and O. Etzioni. Towards adaptive Web sites: Conceptual framework and case study. *Artificial Intelligence*, 118:245–275, 2000.
62. S.M. Ross. *Stochastic Processes*. John Wiley & Sons, New York, NY, 2nd edition, 1996.
63. M. Spielmann. Verification of relational transducers for electronic commerce. In *Proc. 19th ACM Symp. on Principles of Database Systems (PODS’2000)*, pages 92–103, 2000.
64. D.J. Watt. *Six Degrees: The Science of a Connected Age*. William Heinemann, London, 2003.
65. R.J. Wilson. *Introduction to Graph Theory*. Longman, London, 1996.

## **Part I**

---

### **Evolution of Web Structure and Content**



---

## Introduction

This section of the book focusses on the structural and statistical properties of the Web, and how we can take advantage of these in order to understand the Web's evolution and to detect meaningful substructures within it. Since its inception in 1990, the Web has been rapidly growing, and in its current state (in late 2003) its size can be roughly estimated at over four billion pages, with at least an order of magnitude more than this of links.

Chapter 2 discusses the problem of measuring the size of the Web. Measuring the Web's size can help us to identify growth patterns in the Web or subsets of it, and to ascertain the coverage of search engines relative to each other. This in turn may be useful in devising better strategies for digital information management and archiving of Web resources.

Dobra and Fienberg survey previous work in this field and then focus on the work on improving the estimates of search engine coverage carried out by Lawrence and Giles in 1998. Search engines only cover part of the Web, called the indexable Web, which does not include Web material that is hidden from search engines nor Web pages that search engines are not allowed to crawl, for instance, through the robot exclusion protocol. The basic technique used to measure size is the capture–recapture method, which is a well-known statistical technique for measuring wildlife populations. In this case it makes use of the published size of one search engine and its overlap with a second search engine. Assuming that samples taken from both search engines are independent, the size of the indexable Web is estimated as the given size of the first search engine multiplied by the sample size from the second search engine, divided by the size of the overlap of the samples from both engines.

An improvement to the classical capture–recapture technique is suggested via the Rasch model, which relaxes the simplistic assumption that all search engines are independent with respect to the sampling process. Using this model the authors obtain a lower bound of 520 million Web pages for the indexable Web, as it was in 1998, as opposed to Lawrence and Giles's estimate of 320 million. One of the open problems mentioned is how to sample the Web directly rather than through search engine queries.

Chapter 3 examines the problem of identifying Web communities within the graph structure of the Web, where a Web community is composed of a collection of Web pages that are focussed on a particular topic or theme. The focus here is on techniques which make use of the linked structure of the Web rather than the content that can be found in Web pages. Community identification methods can be viewed as a simplification of a clustering task, where the grouping is according to a similarity measure starting from a given seed set of Web pages.

Flake et al. survey previous work on bibliographic metrics, bipartite cores and spectral methods. Bibliographic metrics include coupling and cocitation, which can be adapted to the Web environment to measure structural similarity. The bipartite cores method relies on the intuition that resource lists on a given topic will tend to have overlapping link sets to sites that are focussed on the topic. Spectral methods include Kleinberg's HITS algorithm for identification of hubs and authorities, and Google's PageRank algorithm which can be made topic sensitive by forcing the random walker to jump to focussed pages when a random move is made.

The authors define a Web community as a set of Web pages such that each page within the community has more links to other members of the community than links to pages outside the community. This allows them to recast the problem of community identification in terms of maximum flow methods, which have been widely researched within graph theory. Although in this setting the identification problem is, in general, NP-complete, the authors provide a polynomial-time approximation, which is shown to have some desirable properties. Experimental results for the community algorithm are also presented illustrating its validity. One of the open problems mentioned is that of combining the community algorithm with text mining techniques.

Chapter 4 discusses recent developments in the area of random networks as they relate to the study of the internet and the Web as growing complex networks. This area has become very interdisciplinary, combining work of researchers from a variety of fields, who are all trying to understand and explain various properties of these networks. The seminal study of the Web graph in 2000 by Broder et al. using large-scale crawls of the Web is a milestone in this area, and it has set the scene for much of the later work.

Mendes surveys the area from a statistical physics perspective, concentrating on various characteristics of complex networks such as the degree distribution, the clustering coefficient, shortest paths and the giant component. Using the master equation approach, which has been widely applied by Dorogovtsev and Mendes, the author demonstrates how the evolution of the Web graph can be modelled. For example, regarding the degree distribution of the graph, Mendes shows that a growing network with random uniform attachment of links has an exponential degree distribution, while if the attachment is preferential then the degree distribution is a power law, i.e. the network is scale-free.

Mendes also shows that scale-free networks such as the Web are resilient to random breakdowns but are vulnerable to intentional attacks on highly connected nodes. This has some important implications for the Web. One of the open problems mentioned is that of studying more general models, which unify some of the previous ones that

were studied, in order to understand specific subproperties within the network such as the degree distribution.

Chapter 5 examines quantitative measures that give us insight into the dynamics of the Web, its structure and the quality of its Web pages. The results presented are mainly empirical, based on a study of the Chilean Web carried out during 2000 and 2001. The authors refine the bow-tie structure of the Web discovered by Broder et al. in 2000, and discuss some of their findings relating the age of a Web page and its position within the Web structure. For example, they have found that pages in the main core component of the Web have, on average, higher PageRank values than pages outside the core.

Baeza-Yates et al. survey previous work on modelling the dynamics of Web pages and measuring the rate of change within a page. For estimation purposes, change in Web pages can be modelled as a Poisson process, an application of this being finding an optimal scheduling policy for a Web crawler.

The authors suggest an age-dependent version of PageRank, since PageRank in its original form tends to be biased against newer pages. This bias is intuitive, since new pages are less known and thus are less likely to have many incoming links. One of the open problems mentioned is the analysis of how PageRank changes when some of the incoming links to pages may be removed, in addition to new incoming links being added.

---

# How Large Is the World Wide Web?\*

Adrian Dobra<sup>1</sup> and Stephen E. Fienberg<sup>2</sup>

<sup>1</sup> Institute of Statistics and Decision Sciences, Duke University, Durham NC 27708, USA, [adobra@stat.duke.edu](mailto:adobra@stat.duke.edu)

<sup>2</sup> Department of Statistics and Center for Automated Learning and Discovery, Carnegie Mellon University, Pittsburgh PA 15213, USA, [fienberg@stat.cmu.edu](mailto:fienberg@stat.cmu.edu)

**Summary.** There are many metrics one could consider for estimating the size of the World Wide Web, and in the present chapter we focus on size in terms of the number  $N$  of Web pages. Since a database with all the valid URLs on the Web cannot be constructed and maintained, determining  $N$  by counting is impossible. For the same reasons, estimating  $N$  by directly sampling from the Web is also infeasible. Instead of studying the Web as a whole, one can try to assess the size of the *publicly indexable Web*, which is the part of the Web that is considered for indexing by the major search engines.

Several groups of researchers have invested considerable efforts to develop sound sampling schemes that involve submitting a number of queries to several major search engines. Lawrence and Giles [8] developed a procedure for sampling Web documents by submitting various queries to a number of search engines. We contrast their study with the one performed by Bharat and Broder [2] in November 1997. Although both experiments took place almost in the same period of time, their estimates are significantly different.

In this chapter we review how the size of the indexable Web was estimated by three groups of researchers using three different statistical models: Lawrence and Giles [8, 9], Bharat and Broder [2] and Bradlow and Schmittlein [3]. Then we present a statistical framework for the analysis of data sets collected by query-based sampling, utilizing a hierarchical Bayes formulation of the Rasch model for multiple list population estimation developed in [6]. We explain why this approach seems to be in reasonable accord with the real-world constraints and thus allows us to make credible inferences about the size of the Web. We give two different methods that lead to credible estimates of the size of the Web in a reasonable amount of time and are also consistent with the real-world constraints.

## 1 Introduction

The World Wide Web (henceforth the Web) has become an extremely valuable resource for a wide segment of the world's population. Conventional sources of information (e.g. libraries) have been available to the public for centuries, but the Web has made possible what seemed to be only a researcher's dream: instantaneous access to journals, articles, technical report archives, and other scientific publications. Since

---

\* This chapter is an edited version of a paper presented at Interface 2001 [4].

almost anybody can create and “publish” Web pages, the Web has no coherent structure and consequently it is not easy to establish how much information is available. Evaluation of the size and extent of the Web is difficult not only because of its sheer size, but also because of its dynamic nature. We have to take into account how fast the Web is growing in order to obtain credible estimates of its size. *Growth* in this context is “an amalgam of new Web pages being created, existing Web pages being removed, and existing Web pages being changed” [11]. As a result, any estimate of the size of the Web will be time dependent.

The Web consists of text files written in HyperText Markup Language (HTML). A HTML file contains special fields called *anchor tags*, which allow an author to create a *hyperlink* to another document on the Web. When the user clicks on one of these fields, the Web browser loads the URL specified in the hyperlink, and thus the Web can be seen as a directed graph,  $\mathcal{G}$ , with HTML pages as vertices and hyperlinks as edges. Albert et al. [1] claim that the diameter of  $\mathcal{G}$ , defined as the mean of the number of URLs on the shortest path between any two documents on the Web, can be expressed as a linear function of the number of vertices  $N$  of the graph  $\mathcal{G}$  on a logarithmic scale. Using the value of  $N$  found by Lawrence and Giles [9], they concluded that “two randomly chosen documents on the Web are on average 19 clicks away from each other”. Unfortunately, very little is known about the underlying structure of this highly connected graph. As a consequence, there is no direct method of estimating  $N$ . The dimensions of a database with all possible URLs on the Web will be huge, and, even if we could construct a URL database, we cannot determine which URLs correspond to valid Web documents. Sampling directly from the Web is infeasible: without a list of URLs, known in sample surveys as a *frame*, either implicit or explicit, it is impossible to take a valid probability sample. Alternative methods are also problematic, e.g. the length of the random walks required to generate a distribution over a subset of the Web that is close to the uniform may be extremely large [2].

If we cannot study the Web as a whole, we can try to assess the size of the *publicly indexable Web*. The indexable Web [11] is defined as “the part of the Web which is considered for indexing by the major engines, which excludes pages hidden behind search forms, pages with authorization requirements, etc.”. Search engines such as Northern Light, AltaVista, or HotBot might give the impression that it is very easy to locate any piece of information on the Web. Since “several search engines consistently rank among the top ten sites accessed on the Web” [9], it should be obvious that the search services are used by millions of people daily. However, studies show that the search engines cover “fewer than half the pages available on the Web” [8], and as time goes by, they increasingly fail to keep up with the expanding nature of the Web. Several estimates of the total number of pages [8] indicate that because of the rapid growth of the Web, the fraction of all the valid sites indexed by the search engines continues to decrease.

Search engines have the best Web crawlers, therefore it seems natural that we should try to exploit them. If we want to estimate the portion of the Web covered by the existent search engines, why shouldn’t we use the search engines themselves? Lawrence and Giles [8] developed a procedure for sampling Web documents by

submitting various queries to a number of search engines. We contrast their study with the one performed by Bharat and Broder [2] in November 1997. Although both experiments took place almost in the same period of time, their estimates disagree. In Sect. 2 we show how the size of the Web was estimated by three groups of researchers: Lawrence and Giles [8, 9], Bharat and Broder [2] and Bradlow and Schmittlein [3]. In addition, we explain the discrepancy between the results they obtained. Sect. 3 outlines our approach for calculating a lower bound on the size of the Web based on the data collected by Lawrence and Giles in December 1997. Our objective is to develop a procedure that could be applied in real time, allowing us in the future to monitor the growth of the Web by calculating estimates at several points in time. In the last section we present two different methods that give credible estimates of the size of the Web in a reasonable amount of time and are also consistent with the real-world constraints.

## 2 Web Evaluation

We defined the indexable Web as that part of the Web that is *considered* for indexing by the major engines. We have to make an unequivocal distinction between the indexable Web and the union of the indices of all existent search engines. There are Web documents which might be indexed by a search engine, but, at a fixed time  $t_0$ , were not included in any index. Furthermore, pages with authorization requirements, pages hidden behind search forms, etc., are not compatible with the general architecture of the search engines, and it is unlikely that they will be indexed by any search engine in the near future. We can summarize these ideas as follows:

$$\boxed{\text{Web pages indexed at } t_0} \subset \boxed{\text{Indexable Web at } t_0} \subset \boxed{\text{Entire Web}}$$

where the above inclusions are strict. Based on the inferences we make about the size of the portion of the indexable Web covered by several popular search engines, our goal is to produce an estimate of the size of the whole indexable Web.

Although there is no direct method of counting the number of documents on the Web, the search services disclose the number of documents they have indexed. Unfortunately, counts as reported by the services themselves are not necessarily trustworthy. It is not clear the extent to which duplicate pages, aliased URLs, or pages which no longer exist, are included in the reported counts. Despite these problems, we can still use the self-reported counts as an approximate order of magnitude of the search engines' indices (cf. [8]).

If every single search service has a narrow coverage, the size of any index might offer only a very limited insight about the dimension of the indexable Web. Because any engine has some inherent contribution, the combined coverage of all of the existent engines would allow us to make better inferences about the size of the indexable Web.

## 2.1 Lawrence and Giles Study

Lawrence and Giles [8, 9] developed a procedure for sampling Web pages that does not necessitate access to any confidential database and that can be implemented with fairly modest computational resources using only public query interfaces. This procedure consists of running a number of queries on several search engines and counting the number of results returned by each engine.

Lawrence and Giles [8] studied six major, widely available full-text search engines, namely AltaVista, Infoseek, Excite, HotBot, Lycos and Northern Light. The experimenters selected 575 queries issued by scientists at the NEC Research Institute between 15 and 17 December 1997 and submitted those queries to the six search services. They retrieved all of the results accessible through every engine and for each document they recorded the search engines which were able to locate it. Lawrence and Giles implemented a number of reliability assessments because the data obtained in this way cannot be used as-is due to several experimental sources of bias.

Since search engines do not update their databases frequently enough, they often return URLs relating to Web documents that no longer exist or that may have changed and are no longer relevant to the query. Lawrence and Giles retained only those documents that could be downloaded, and then they removed duplicates including identical pages with different URLs. Some engines are case-sensitive and some are not, hence Lawrence and Giles did not use queries that contained uppercase characters.

Another potential problem is that the six search engines use various ranking algorithms to assess relevance. The Web documents served up by an engine should be perceived as the “best” matches as determined by the ranking procedure. Since relevance is difficult to determine without actually viewing the pages, ranking algorithms might seriously bias our findings. To prevent this from happening, Lawrence and Giles retained only the queries for which they were able to examine the entire set of results. Queries returning more than 600 documents (from all engines combined after the removal of duplicates) were discarded for the purposes of the analysis.

Lawrence and Giles provided us with data in the form of a  $575 \times 63$  matrix. Each row contains the counts for an individual query. The first six columns are the number of pages found by AltaVista, Infoseek, Excite, etc. The next columns are the number of pages found by any two engines, then the pages found by any three engines, and so on. The last column contains the number of pages found by all six engines. Using the principle of inclusion and exclusion, we transformed the “raw” data into 575 cross-classifying tables of dimension  $2^6 - 1$ . Let  $X_1, X_2, \dots, X_6$  be categorical variables corresponding to AltaVista, Infoseek, Excite, HotBot, Lycos and Northern Light, respectively. Each variable has two levels: “1” stands for “found page” and “0” stands for “not found”. Let  $\mathcal{D}$  denote the set of all binary vectors of length 6. The contingency table for query  $k$  ( $1 \leq k \leq 575$ ) can be expressed as  $\mathcal{S}_k = \{r_s^k | s \in \mathcal{D}\}$ . For a given query, we do not know how many pages were *not* found by all six engines, therefore all 575 tables have a missing cell, which can be interpreted as the difference between the “real” number of pages existing on the Web and the number of pages actually found by the six engines for the queries in question.

**Table 1.** Multiple list data for query 140 (S. Lawrence and C.L. Giles, private communication)

				Northern Light							
				yes				no			
				Lycos				Lycos			
				yes		no		yes		no	
				HotBot	HotBot	HotBot	HotBot	HotBot	HotBot	HotBot	HotBot
				yes	no	yes	no	yes	no	yes	no
AltaVista	yes	Excite	yes	1	0	2	0	0	0	1	0
			no	2	0	3	2	0	0	0	2
	yes	Infoseek	yes	1	0	2	1	0	0	3	4
			no	1	3	0	8	2	0	3	19
	no	Excite	yes	0	0	0	1	0	0	0	0
			no	0	0	1	1	0	0	5	4
	no	Infoseek	yes	0	0	0	1	0	0	4	22
			no	0	0	7	17	2	3	31	?

The quality of the analysis we want to perform depends on other factors which might or might not turn out to be significant. Web documents were added, removed, edited, and modified while the experimenters collected the data, hence the search results might also change. Search engines first look for the best matches within the segment of their index loaded in the main memory and only if the matches they found are not satisfactory, they expand the search to the rest of the database. This means that if we would submit the same query to the same search service at different times of the day, the set of results fetched might not be the same. Under some (nearly) improbable circumstances, a search engine might not return any documents present in other search engines' databases, in which case we would not be able to estimate the overlap between indices.

We examined the intersections between the sets of pages corresponding to the 575 queries aggregated over the six search engines and concluded that all the interactions between two queries appear to be significantly smaller than any set of pages matching a query. The segments of the Web defined by the 575 queries are for all practical purposes disjoint, hence we can view our data as a seven-way contingency table  $\mathcal{S}$  in which "query" is a multilevel stratifying variable.

We determined which engine performs better than the others. The *relative coverage* [11] of a search engine is defined by the number of references returned by a search service divided by the total number of distinct references returned. Notice that we can compute this ratio without having to estimate the missing cell. Table 2 shows our calculations of the relative coverage for the six search services considered. HotBot appears to have the largest coverage, followed by AltaVista and Northern Light. Although relative coverage can express the quality of a search service with respect to the others, it cannot be used if we want to find a way to measure the *combined* coverage of the six search engines with respect to the entire indexable Web.



By analyzing the overlap between pairs of search engines, one can easily calculate the fraction of the indexable Web covered by any of the six search engines. Since HotBot had reportedly indexed 110 million pages as of December 1997, Lawrence and Giles estimated that the absolute size of the indexable Web should be roughly 320 million pages. We discuss in detail the validity of their estimate in Sect. 2.4 as part of our reanalysis of their data. In February 1999, Lawrence and Giles [9] repeated

**Table 2.** Estimated relative coverage of the six search engines employed

Service	Coverage
HotBot	52.02%
AltaVista	37.23%
Northern Light	26.39%
Excite	19.11%
InfoSeek	13.24%
Lycos	4.15%

their experiment. The number of search engines was increased to 11 (AltaVista, Euroseek, Excite, Google, HotBot, Infoseek, Lycos, Microsoft, Northern Light, Snap and Yahoo) and the number of queries was expanded to 1,050, hence the data this time consists of a  $1,050 \times (2^{11} - 1)$  array. The experimenters did not make clear whether the 575 queries used for the first study were among the 1,050 queries used for the second one. Northern Light had indexed 128 million pages at the time of the experiments, hence Lawrence and Giles approximated that there were 800 million pages on the indexable Web. The estimation method was similar to the one employed in the previous study. Unfortunately, their analysis was done dynamically and the new data were not retained for possible reanalyses.

## 2.2 Bharat and Broder Study

In November 1997, Bharat and Broder [2] performed an analysis analogous in many respects to the one carried out by Lawrence and Giles. They employed only four engines, i.e. AltaVista, Excite, Infoseek and HotBot. Instead of measuring directly the sizes and overlaps of the four search services, their approach involved generating random URLs from the database of a particular search engine and checking whether these pages were also indexed by the other search services.

The experimenters approximated sampling and checking through queries. Rather than choosing queries made by real users, Bharat and Broder randomly generated their own queries. The queries were derived from a lexicon of about 400,000 words built from 300,000 documents existing in the Yahoo! hierarchy. The artificially generated queries were presented to one search service and the search results were retrieved. Since it is very hard to get a hold of the entire set of results, Bharat and Broder picked a URL at random from the top 100 matches that were found for every query, hence the results were heavily dependent on the ranking algorithm used by every search

engine and also on the particular choice of lexicon. Both the ranking strategy and the lexicon can introduce serious experimental bias, that is, some documents will have better chances of being included in the sample than others.

For every query selected from one of the four indices, Bharat and Broder created a *strong query* intended to uniquely identify that particular page. They built the strong query by picking the most significant terms on the page and submitted it to the other search services. An engine  $\mathcal{E}$  had indexed page  $\mathcal{P}$  if  $\mathcal{P}$  was present in the set of results fetched from  $\mathcal{E}$ . Because there is so much duplication on the Web, the set of results obtained might contain more than one document. It is not clear whether  $\mathcal{E}$  would have found page  $\mathcal{P}$  if the original query that generated  $\mathcal{P}$  had been submitted to  $\mathcal{E}$ .

Bharat and Broder performed two series of experiments: trials 1 (10,000 disjunctive queries) and 2 (5,000 conjunctive queries) in mid-1997, and trials 3 (10,000 disjunctive queries) and 4 (10,000 conjunctive queries) in November 1997. We can see that the set of queries employed was considerably larger than the set used by Lawrence and Giles [8].

A more elaborate method than the one used by Lawrence and Giles [8, 9] was employed to assess what fraction of the indexable Web was covered by an individual search engine involved in the study. The experimenters calculated engine size estimates by minimizing the sum of the squared differences of the estimated overlaps between pairs of search engines. Since AltaVista reportedly indexed 100 million pages, Bharat and Broder concluded that the indexable Web had roughly 160 million pages in November 1997. We will come back with a detailed discussion of the validity of these results in Sect. 2.4.

### 2.3 Bradlow and Schmittlein Study

Another attempt to evaluate the Web was carried out by Bradlow and Schmittlein [3] during October 1998. They tried to assess the capability of six search engines (the very same engines employed in Lawrence and Giles [8]) to find marketing and managerial information using query-based sampling. Twenty phrases were chosen to be submitted to the search engines. The phrases had to be representative of the marketing world and also adequately precise (any number of pages could be relevant for an ambiguous query, hence our inferences could be adversely biased if too many relevant pages were found).

The six search engines combined returned 1,588 different pages. For each of these pages, the experimenters recorded the binary pattern of length 6 describing what engines successfully detected the page (as before, “1” stands for “found page” and “0” for “not found”), the number of page links (0 – 5, 6 – 10, or 10+) and the domain type, indicating whether the site where the page was located was commercial (.com), academic (.edu), an organization (.org) or some other type of site (“other”). In addition, two phrase characteristics were also recorded – newer versus older and academic versus managerial – for more details see Bradlow and Schmittlein [3].

The originality of this approach comes in the way Bradlow and Schmittlein analyzed the data they collected. Each search engine and Web page are assumed to lie in a  $D$ -dimensional space. The probability that a given engine will capture some page is

a decreasing function of the distance between the engine and the page, hence a search engine is more likely to capture pages located in its immediate vicinity than pages that are situated at some considerable distance.

In the first model they proposed, they placed all the engines in the origin of a one-dimensional space ( $D = 1$ ). Search engines tend to find the same Web pages, and consequently the “less resourceful” engines index only a subset of the pages indexed by the “more powerful” engines. The second model studied differs from the first one only with respect to the number of hypothesized dimensions of the underlying space; they took  $D = 2$  to be a reasonable choice. Their third model is more flexible than the previous two because it allows the engine locations to vary in a two-dimensional space.

To be more specific, let  $p_{ijk}$  be the probability that the  $k$ th URL for the  $j$ th phrase is found by engine  $i$ . Moreover,  $d_{ijk}$  denotes a squared Mahalanobis distance between the location of the  $i$ th engine and the location of the  $k$ th URL for phrase  $j$  in the  $D$ -dimensional space. If  $u$  is “the rate at which the probability an engine finds a given URL drops off”, we can express  $p_{ijk}$  as a function of  $d_{ijk}$  by

$$p_{ijk} = \frac{1}{1 + d_{ijk}^u}. \quad (1)$$

Bradlow and Schmittlein fit all three models using a Markov chain Monte Carlo sampler. The first two models were invalidated by the data, while the third seems to fit their data reasonably well. This is a clear indication that every search engine “carves out” its own location in the URL space.

Bradlow and Schmittlein [3] conclude that, for marketing/managerial queries, “the reader should feel confident that the search engines cover about 90% of what exists to be found for these kind of phrases”. Although the authors argue that their modeling technique is superior to any other study performed and that “these kinds of marketing/management documents are relatively easy to locate”, the result they came up with appears to conflict with what we know about search engine behavior. There are elements of their model and analyses, however, that would be worth further investigation as elaborations of the approach suggested in this chapter.

## 2.4 The Size of the Indexable Web

Here we describe explicitly the statistical models and inherent assumptions that underlie the estimates of Lawrence and Giles [8, 9], and Bharat and Broder [2].

Let  $\mathcal{E}_1$  and  $\mathcal{E}_2$  be two search engines with indices  $E_1$  and  $E_2$ , respectively. Denote by  $\Omega$  the complete set of documents available on the indexable Web. We make two major assumptions:

- (A1) The indices  $E_1$  and  $E_2$  are samples drawn from a uniform distribution over  $\Omega$ .
- (A2)  $E_1$  and  $E_2$  are independent.

Denote by  $|A|$  the number of elements of the set  $A$ . The first assumption says that

$$|\Omega| = \frac{|E_1|}{P(E_1)}, \quad (2)$$

while (A2) implies

$$P(E_1) = P(E_1 \cap E_2 | E_2). \quad (3)$$

We can estimate  $P(E_1 \cap E_2 | E_2)$  from our data by

$$\hat{P}(E_1 \cap E_2 | E_2) = \frac{|A_1 \cap A_2|}{|A_2|}, \quad (4)$$

where  $A_1$  and  $A_2$  are the sets of pages returned when all the queries utilized in a study were submitted to engines  $\mathcal{E}_1$  and  $\mathcal{E}_2$ . As a result, the size of the Web can be estimated by

$$|\widehat{\Omega}| = \left\lfloor \frac{|E_1| \cdot |A_2|}{|A_1 \cap A_2|} \right\rfloor, \quad (5)$$

where  $\lfloor x \rfloor$  is the largest integer smaller than or equal to  $x$ .

Formula (5) gives us a way to extrapolate the size of the indexable Web based on the published size of the index of an engine  $\mathcal{E}_1$  and on the estimated overlap between  $\mathcal{E}_1$  and another engine  $\mathcal{E}_2$ . But assumptions (A1)–(A2) are not necessarily satisfied. The search engines do not index Web documents at random. They employ two major techniques to detect new pages: user registration and following (hyper)links [9]. On one hand, people who publish on the Web have the tendency to register their pages with as many services as possible. On the other hand, popular pages that have more links to them will have greater chances to be indexed than new (hence unlinked) pages. We infer that search engines will be more inclined to index several well-defined fractions of the indexable Web, which will induce a positive or negative correlation between any two search engines indices. Since the probability of a page being indexed is not constant, a search engine's index will represent a biased sample from the entire population of Web documents.

The estimate in Lawrence and Giles [8] was based on the overlap between AltaVista and HotBot. Since they were the engines with the largest (relative) coverage at the time of the tests (among the six engines studied), their indices will have “lower dependence because they can index more pages other than the pages the users register and they can index more of the less popular pages on the Web” [8]. The reported size of HotBot was 110 million pages, hence Lawrence and Giles found 320 million pages to be an estimate of the size of the indexable Web in December 1997. Bharat and Broder argue that the indexable Web should have about 160 million pages as of November 1997, since AltaVista had reportedly indexed 100 million pages at that time and “had indexed an estimated 62% of the combined set of URLs” [2]. There is a clear discrepancy between the two estimates. Since the queries used by Lawrence and Giles were issued by researchers, they relate to topics few users search for. Search engines are oriented toward finding information the average user wants, thus Lawrence

and Giles might have underestimated the overlap between indices. On the other hand, Bharat and Broder might have overestimated the overlap since the engines have a tendency to locate content-rich documents and these are the documents the randomly generated queries are inclined to match. As a consequence, it appears that Lawrence and Giles overestimated the size of the indexable Web, whereas Bharat and Broder underestimated it.

Although the Web is a dynamic environment, it can be assumed that the population of Web documents is *closed* at a fixed time  $t_0$ , i.e. “there are no changes in the size of the population due to birth, death, emigration or immigration from one sample to the next” [5]. This definition translates in our framework to: *no Web pages were added, deleted or modified while the data was collected*. Since the entire indexable Web can be considered closed at a fixed time  $t_0$ , the subpopulation of pages that would match query  $k$ ,  $1 \leq k \leq 575$ , will also be closed at time  $t_0$ . Our goal is to assess the size  $N_k$  of the population of pages defined by query  $k$  at time  $t_0$  (i.e., December 1997) using the standard multiple-recapture approach to population estimation. In the capture–recapture terminology, Web pages are referred to as *individuals* or *objects* and search engines as *lists*.

More precisely, we have six samples  $L_1^k, \dots, L_6^k$ , where  $L_i^k$ ,  $1 \leq i \leq 6$ , represents the best matches for query  $k$  found by engine  $i$ . Following Fienberg [5], let  $n_{1+}^k$  and  $n_{+1}^k$  be the number of individuals in the samples  $L_1^k$  and  $L_2^k$ , respectively, and  $n_{11}^k$  be the number of individuals in both lists. The classical capture–recapture estimate for  $N_k$  based on the first two lists is

$$\hat{N}_k = \left\lfloor \frac{n_{1+}^k \cdot n_{+1}^k}{n_{11}^k} \right\rfloor, \quad (6)$$

i.e. the traditional “Petersen” estimate. We can compute the Petersen estimates for  $N_k$  based on all pairs of the six available lists. The Petersen estimate assumes the objects are heterogeneous (A1) and the lists are pairwise independent (A2), hence Eq. (5) and the Petersen estimate (6) are built on the same suppositions. Moreover, we can see that the estimate Lawrence and Giles found for the indexable Web is nothing more than a Petersen estimate scaled up by a factor, namely the number of pages HotBot had reportedly indexed divided by the total number of pages found by HotBot for the 575 queries.

We considered the seven-way table  $\mathcal{S}$  collapsed across queries and computed the traditional capture–recapture estimates for the number  $\mathcal{N}$  of Web pages matching at least one of the queries used. Only 7 out of 15 were above the observed number of objects in the six lists, which represents a lower bound for the “real” number of pages  $\mathcal{N}$ . In Fig. 1, we give the proportion  $\mathcal{T}_k$  of Petersen estimates smaller than the observed number of pages  $n_k$  for every six-way contingency table  $\mathcal{S}_k$ . Proportion  $\mathcal{T}_k$  is bigger than 50% for almost half the queries. This is clear evidence that the assumptions (A1)–(A2) do not hold. These calculations also suggest that there is positive and also negative dependence between pairs of search engines across the 575 queries. Since the six search engines attempt to maintain full-text indices of the entire

indexable Web, the interactions we observed are the result of the bias introduced by the query-based sampling.

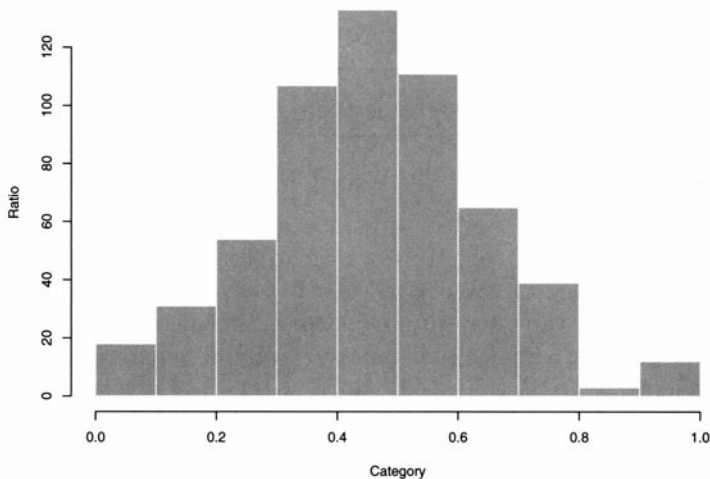


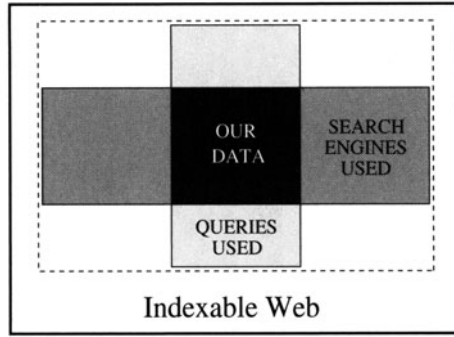
Fig. 1. Proportion of Petersen estimates smaller than  $n_k$

### 3 Our Approach for Estimating the Size of the Web

The 575 queries of Lawrence and Giles [8] define a population of Web documents, while the union of the indices of the search engines employed define another population of pages. We observed the intersection between the two populations and summarized it in a seven-way contingency table  $\mathcal{S}$  with missing entries. If we were able to approximate the number of pages *not* found by all the search engines used for every query, we could draw inferences about the dimension of the population of pages relevant to the 575 queries. Based on this estimate and on the published size of the index of one of the six search engines, we could extrapolate the number of Web documents contained in the dotted rectangle in Fig. 2. This approach provides a lower bound of the size of the indexable Web as of December 1997, and in the following sections we propose one possible implementation of it using an approach suggested in Fienberg et al. [6].

#### 3.1 The Rasch Model

The subpopulation of Web documents matching query  $k$ ,  $1 \leq k \leq 575$ , is a closed population at a given point in time. Our objective is to estimate its unknown size



**Fig. 2.** The two populations of Web pages that define the observed data

$N_k$  using multiple lists or sources. We make use of the  $k$ th contingency table  $S_k$ , that cross-classifies individuals (Web pages) based on which search engines (or lists) were able to locate them. This is the usual setting for the multiple-recapture population estimation problem, which originated in estimating wildlife and fish populations.

Let  $i = 1, \dots, N_k$  index the individuals and  $j = 1, \dots, J = 6$  index the lists. Define

$$X_{ij} = \begin{cases} 1, & \text{if engine } j \text{ located page } i, \\ 0, & \text{otherwise.} \end{cases}$$

In other words,  $X_{ij} = 1$  if individual  $i$  appears on list  $j$ . Let  $p_{ij}$  be the probability of this event. The number of Web pages identified by at least one search engine for query  $k$  is  $n_k$ . Clearly, estimating  $N_k$  is equivalent to estimating  $N_k - n_k$ . We require a model that allows for

1. **Heterogeneity of capture probabilities:** The probability of a page being indexed by a search engine is not constant. Pages with more links to them are more likely to be located by a search service [9].
2. **List dependencies:** Search engines are more inclined to index certain fractions of the Web, hence the search results they return will be correlated.
3. **Heterogeneity among search services:** Each engine has a specific built-in searching mechanism and because this mechanism is different from one engine to the other, the set of Web documents indexed by every service will also be different.

Rasch [10] introduced a simple mixed-effects generalized linear model that allows for object heterogeneity and list heterogeneity. The multiple-recapture model can be expressed as

$$\log \left\{ \frac{p_{ij}}{1 - p_{ij}} \right\} = \theta_i + \beta_j; \quad i = 1, \dots, N_k; \quad j = 1, \dots, J; \quad (7)$$

where  $\theta_i$  is the random catchability effect for the  $i$ th Web page, and  $\beta_j$  is the fixed effect for the penetration of engine  $j$  into the target population represented by all indexable Web documents relevant for the  $k$ th query. The heterogeneity of capture probabilities across objects depends on the distribution  $F_\Theta$  of  $\theta = (\theta_1, \dots, \theta_{N_k})$ . Note that, if we set the  $\theta_i$  in Eq. (7) equal to zero, the log-odds of inclusion of object  $i$  on list  $j$  depends only on the list, and thus the Rasch model reduces to the traditional capture-recapture model with independent lists. When the  $\theta_i$ 's are different from zero and we treat them as random effects, this model is multilevel, with lists at one level and individuals at another.

Fienberg et al. [6] showed how to analyze one query using a Bayesian approach for estimating the parameters of the Rasch model. They employed the following full Bayesian specification:

$$\begin{cases} X_{ij} & \sim \text{Bern}(p_j|\theta_i); & i = 1, \dots, N_k; j = 1, \dots, J; \\ \theta_i & \sim F_\Theta(\theta_i); & i = 1, \dots, N_k; \\ \beta_j & \sim G_\beta(\beta_j); & j = 1, \dots, J. \end{cases} \quad (8)$$

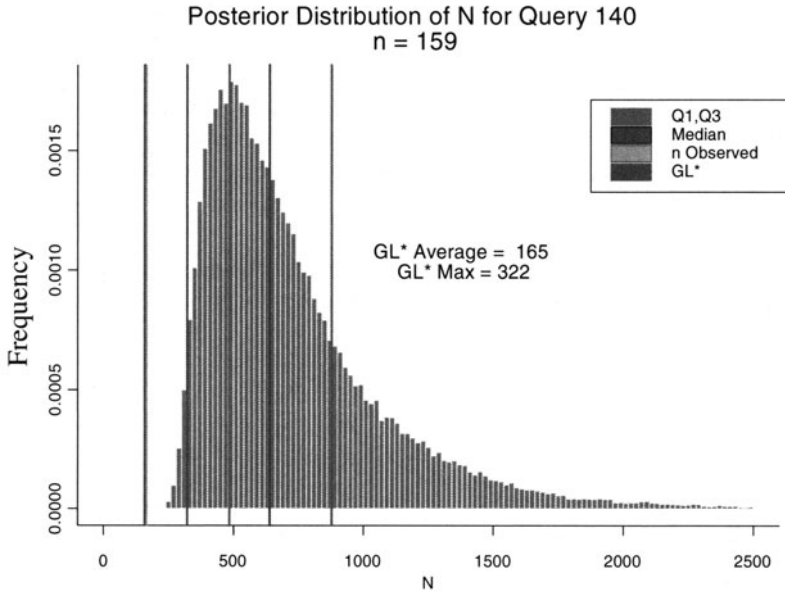
This permits us to describe all of the model components, and alter precisely those parts that need adjustment to reflect the dependency in the data. We use an extension of the Markov chain Monte Carlo technique for fitting item-response models, as described in Johnson et al. [7]. Following Fienberg et al. [6], we assume that the vector of list parameters  $\beta = (\beta_1, \dots, \beta_J)$  is distributed  $\mathbf{N}_6(\mathbf{0}, 10 \cdot \mathbf{I}_6)$  and is independent of  $(\theta, \sigma^2, N_k)$ . The catchability parameter vector is distributed  $\theta|\sigma^2, N_k \sim \mathbf{N}_{N_k}(\mathbf{0}, \sigma^2 \cdot \mathbf{I})$  and  $\sigma^2 \sim \Gamma^{-1}(1, 1)$ . This distribution is proper but presumes that we have little knowledge about the search engines indices and about their underlying indexing algorithms. As the prior distribution of  $N_k$ , we use a variation of the Jeffrey's prior

$$f_{\mathbf{N}_k}(N_k) \propto \frac{1}{N_k} \cdot I_{\{n_k < N_k < N_k^{\max}\}}. \quad (9)$$

This specification is robust to the choice of  $N_k^{\max}$  and can be as small as 10,000 or as large as 1,500,000. The latter threshold was used when fitting the Rasch model for the table collapsed across queries.

To illustrate the use of this model, we consider query 140, which has  $n_{140} = 159$  URLs, and we compare the results obtained by fitting the Bayesian Rasch model with the classical Petersen estimates. The posterior distribution of  $N_{140}$  is skewed, while the median (639) is not very close to the mode (481; Fig. 3). The 95% confidence interval for  $N_{140}$  is [330, 1691]. The 95% *highest posterior density* (HPD henceforth) interval is [268, 1450]. This 95% HPD interval for the Bayesian Rasch model is an equal-tailed probability interval [6]. The lower end of the HPD interval is only slightly smaller than the lower end of the 95% confidence interval, whereas the upper ends of the two intervals are a lot farther apart. We are not surprised by this fact since the posterior distribution has a long right tail. Both the mean (165) and the maximum (322) of the Petersen estimates are bigger than the observed number of pages. The Petersen estimates suggest that the expected number of pages is only twice as large as





**Fig. 3.** Posterior distribution of the projected number of Web pages  $N_k$  for query  $Q_{140}$

$n_{140}$ , as compared with the Rasch model estimate, which is at least four times larger than  $n_{140}$ .

As the observed number of pages  $n_k$  increases, the posterior distribution of the projected number of pages  $N_k$  moves toward a symmetric distribution. The Petersen estimator constantly underestimates  $N_k$  when compared with the inferences we draw through the Rasch model. Since the assumptions the Rasch model is built on are a lot closer to reality than the assumptions (A1)–(A2) we make when using the Petersen estimator, we are inclined to give more credit to the Rasch model. Moreover, the Lawrence and Giles approximation of the size of the indexable Web is of the same order of magnitude as the Petersen estimator.

### 3.2 Collapsing versus Regression

Estimating the total number of documents  $\mathcal{N}$  on the indexable Web relevant to at least one of the 575 queries is a key step in our analysis. Each of the 575 queries defines approximately disjoint segments of the Web. Since the Rasch model provides a good estimate of the size  $\hat{N}_k$  of each subpopulation, we would be tempted to approximate  $\mathcal{N}$  as

$$\hat{\mathcal{N}} = \sum_{k=1}^{575} \hat{N}_k. \quad (10)$$

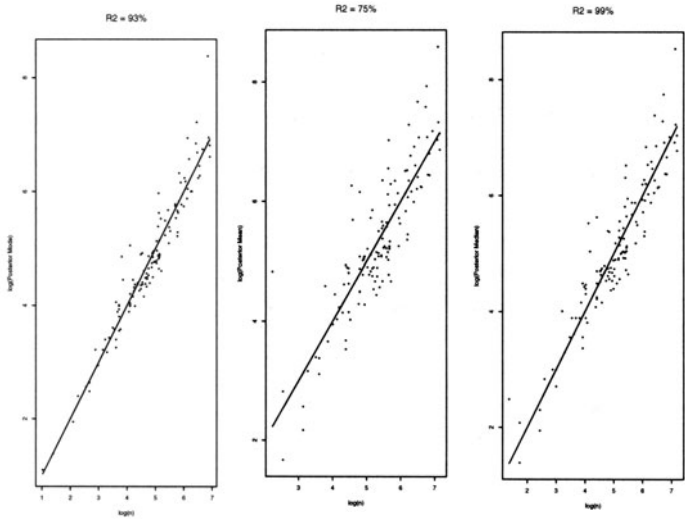
Although simple and appealing, it is not easy to make use of Eq. (10) in practice since it requires fitting a Rasch model for every query we work with. An alternative solution is to fit the Rasch model for the contingency table  $\mathcal{S}_0$  derived from the seven-way table  $\mathcal{S}$  by collapsing across queries. We are aware that the different queries induce heterogeneous populations of pages, hence building our reasoning solely on the six-way cross-classification  $\mathcal{S}_0$  might seriously bias our findings. On the other hand, the heterogeneity effect might not be as strong as we expect and so it might be adequate to make use of  $\mathcal{S}_0$ .

To account for the possible heterogeneity effect, we sampled without replacement 128 queries from the 575 queries (about 20%) contained in the data set, and we used the Rasch model to estimate the number of relevant pages that were not found by any of the search engines. We estimated  $N_k$  for the queries not selected in the sample by employing simple linear regression. We modeled the posterior mean, median and mode of  $N_k$  as a function of the observed number of pages for every query in the sample and ended up with the following models:

$$\begin{aligned} \text{(M1)} \quad \log(\hat{N}_k^{\text{mean}}) &= 4.67 + 19.2 \cdot \log(n_k), \\ \text{(M2)} \quad \log(\hat{N}_k^{\text{median}}) &= 130.6 \cdot \log(n_k), \\ \text{(M3)} \quad \log(\hat{N}_k^{\text{mode}}) &= -2.88 + 41.94 \cdot \log(n_k). \end{aligned} \quad (11)$$

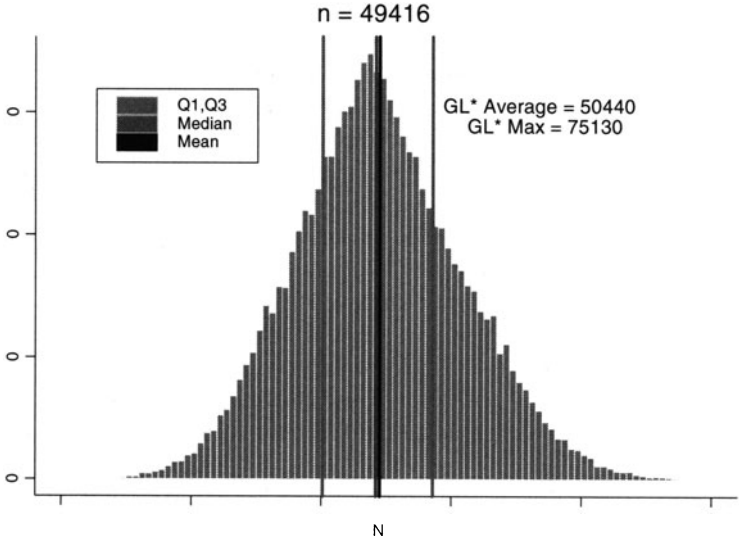
The coefficients of determination for models (M1), (M2) and (M3) are 75%, 99% and 93% respectively. Care should be taken when interpreting the coefficient of determination  $R^2$  for (M2), since the intercept is not present in the model. The plot of observed versus fitted values (Fig. 4) confirms the validity of the models we proposed. It appears that the projected number of pages germane to query  $\mathcal{Q}_k$  is directly proportional on a logarithmic scale to  $n_k$ , that is, the total number of Web pages identified by the six search engines combined. The models (M2) and (M3), which are the “best” regressions, can be employed to predict  $N_k$  for the queries for which we did not fit the Rasch models. The six search engines employed by Lawrence and Giles [8] identified 49,416 pages on the Web relevant to at least one of the 575 queries. The predicted number of relevant pages is 167,298 if we use model (M2) and 125,368 if we use model (M3). Therefore these regression-based projections suggest that there exist *at least* twice as many relevant pages on the Web that were not found by any search engine.

In Fig. 5, we present the posterior distribution of  $\mathcal{N}$  from fitting the Bayesian Rasch model for table  $\mathcal{S}_0$ . This distribution is symmetric and unimodal, with a posterior median equal to  $N_0 = 184,160$ . The 95% HPD interval for  $\mathcal{N}$  is [173427, 199939]. The mean 50,440 of the Petersen estimates is only slightly bigger than the total number of pages  $n_0 = 49,416$  captured by the combined search engines for all the queries, whereas the maximum is 75,130. Consequently, the projected number of pages  $\hat{\mathcal{N}}$  using the Petersen estimator is not even twice as large as  $n_0$ , while using the Rasch model the same quantity would be approximated to be almost four times as large as  $n_0$ .



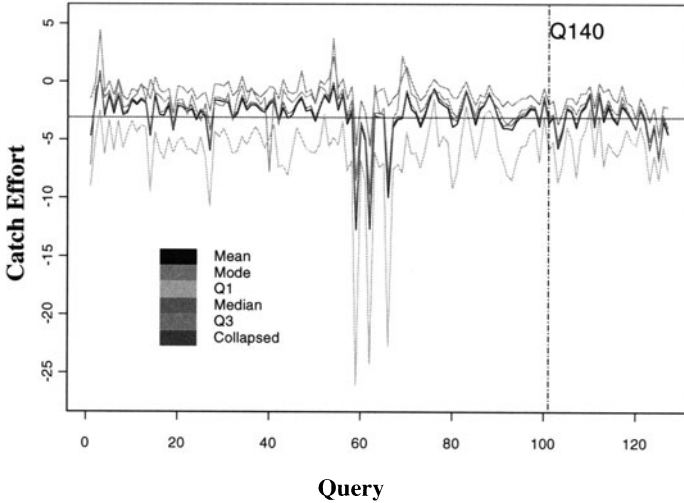
**Fig. 4.** Regression of the posterior mode, mean and median of  $N_k$  on the observed number of pages

Posterior Distribution of  $N$  for the Table Collapsed Across Queries



**Fig. 5.** Posterior distribution of  $N$  for the seven-way table  $S$  aggregated across queries

Recall that  $\beta_j$ ,  $1 \leq j \leq 6$ , is the fixed effect for the penetration of engine  $j$  into the target population. Figure 6 portrays the catch effort of AltaVista across all of the 128 sample queries. We plotted several summary statistics based on the posterior distribution of  $\beta_1$  from the samples we generated using the Rasch model. The overall catch effort  $\beta_1^0$  of AltaVista is taken to be the posterior median of the Rasch model for table  $S_0$ . Figure 6 offers unmistakable evidence that the performance of AltaVista remains stable across the 575 queries used, since  $\beta_1^0$  stays within the 95% confidence limits for almost all 128 queries. The posterior distributions of those  $\beta_1$ 's for which  $\beta_1^0$  lies outside the 95% confidence intervals might not be well approximated due to insufficient information – few Web pages observed for the corresponding queries. The rest of the search engines exhibit the same unvarying behavior. When interpreting Fig. 6, we have to keep in mind that the 575 queries have no “natural” order; they were labeled with “1”, “2”, ..., “575” in the same order in which Lawrence and Giles [8] included them in the initial  $575 \times 63$  matrix they provided us with. This means that the curves in Fig. 6 have no intrinsic meaning. However, Fig. 6 is useful for observing the tightness of the quantiles of the estimated catch efforts of the queries about the overall catch effort: only 3 queries of the 128 deviate significantly from the collapsed value.



**Fig. 6.** Catchability effect of AltaVista across the queries selected in the sample. The vertical axis represents the number value of  $\beta_1$  in model (7) and the curves connect the quantiles of the 128 sample queries displayed along the horizontal axis in an arbitrary order

## 4 Scaling Up to the Web

We are now in a position to provide estimates for  $\mathcal{N}$  based on the analyses described in the preceding section:

- **Method 1:** Select a sample from the set of queries, fit the Rasch model for every sample query and extend the results to the rest of the queries via regression.
- **Method 2:** Find a direct estimate for  $\mathcal{N}$  by fitting the Rasch model for the seven-way table  $\mathcal{S}$  collapsed across queries.

For the Lawrence and Giles data, method 1 gives  $\hat{\mathcal{N}}_1 = 167,298$  as an estimate for  $\mathcal{N}$  if model (M2) is employed, while using method 2 we obtain a slightly larger value, namely  $\hat{\mathcal{N}}_2 = 184,160$ . Thus both techniques return results within the same order of magnitude. However, Method 2 fully overlooks the heterogeneity existent among queries and although this method is less expensive to implement, in some particular circumstances we might favor method 1.

Table 3 gives the estimates of the absolute coverage of the six search engines we obtained by employing methods 1 and 2. We contrast our findings with the coverage estimates of Lawrence and Giles [8]. Our estimates suggest that HotBot, the engine with the largest coverage in December 1997, indexed only about 15% of the indexable Web, rather than 34% as calculated by Lawrence and Giles. In addition, our combined coverage of the six search engines is approximately equal to the coverage of AltaVista estimated by Lawrence and Giles!

**Table 3.** Estimated coverage of the search engines used relative to the indexable Web as of December 1997 (Percentages)

	Estimates based on		Lawrence and Giles [8]
	Method 1	Method 2	estimates
Combined coverage of engines used	29.54	27.00	—
AltaVista	11.00	10.00	28.00
Infoseek	3.91	3.60	10.00
Excite	5.65	5.12	14.00
HotBot	15.37	14.00	34.00
Lycos	1.23	1.11	3.00
Northern Light	7.80	7.00	20.00
Common coverage of engines used	0.06	0.03	—

We cannot make inferences about the size of the indexable Web based on our data alone. Consider a search engine  $\mathcal{E}_1$  with index  $E_1$ . The relationship in Eq. (2) tells us that the number of documents available on the indexable Web can be estimated by

$$\left\lfloor \frac{|E_1|}{P(E_1)} \right\rfloor. \quad (12)$$

We approximate  $P(E_1)$  as the ratio between the total number of pages located by  $\mathcal{E}_1$  for all queries used and the estimate for  $\mathcal{N}$  we employed. Currently, we have no choice but to rely on the size of the index of  $\mathcal{E}_1$  as reported by the engine itself. Since these published estimates are not reliable, we used Eq. (12) for several search engines and compared the results we obtained (Table 4). Lawrence and Giles argued that HotBot had reportedly indexed 110 million pages as of December 1997, and consequently they based their estimates on this value. On the other hand, Bharat and Broder [2] claimed that “Search Engine Watch reported the following search engines sizes (as of November 5, 1997): AltaVista = 100 million pages, HotBot = 80 million, Excite = 55 million, and Infoseek = 30 million pages”. The first row uses  $\hat{\mathcal{N}}_1$ , while all the other values use  $\hat{\mathcal{N}}_2$  as an estimate of  $\mathcal{N}$ .

**Table 4.** Absolute estimates for the size of the Web as of December 1997 (millions of pages)

	Reported Sizes				
	HotBot (80)	HotBot* (110)	Infoseek (30)	AltaVista (100)	Excite (55)
Our Web size (Method 1)	520.63	715.87	767.30	909.29	974.21
Combined coverage of engines used (collapsed)	153.78	211.45	226.46	268.57	287.76
AltaVista (collapsed)	57.26	78.73	84.39	100.00	107.15
Infoseek (collapsed)	20.36	27.99	30.00	35.55	38.09
Excite (collapsed)	29.39	40.42	43.32	51.33	55.00
HotBot (collapsed)	80.00	110.00	117.90	139.71	149.70
HotBot (collapsed)	80.00	110.00	117.90	139.71	149.70
Lycos (collapsed)	6.39	8.78	9.42	11.16	11.96
Northern Light (collapsed)	40.58	55.80	59.81	70.88	75.94
Common coverage of engines used (collapsed)	0.16	0.22	0.23	0.28	0.30

Our lowest bound of the size of the indexable Web is 520 million pages, while Lawrence and Giles [8] obtained an estimate of 320 million pages as of December 1997. Remember that Bharat and Broder [2] argued that the Web had only 200 million pages in November 1997. In order to contrast our inferences with the results found by Lawrence and Giles, we scaled up the posterior distribution of  $\mathcal{N}$  from fitting the Rasch model for table  $\mathcal{S}_0$ , using an estimate of the size of HotBot of 110 million pages. This technique allows us to find a distribution of the number of pages available on the indexable Web. The median of this distribution is 788 million pages (see Table 4), while the 95% HPD interval is [742, 856] million pages. If we use the same “external”

information as Lawrence and Giles, we would say that the Web was at least twice as big in 1997 as what was believed until today [2, 8]. In addition, HotBot seems to have the largest index, between 80 and 150 million pages, followed by AltaVista, between 57 and 107 million pages.

We have to emphasize that the method we used for assessing the size of the Web has several shortcomings, and consequently we need to be very careful when interpreting the results obtained by employing it. We pointed out before that the reported sizes of search engines indices are far from being reliable, hence the quantity with which we scale up might not reflect the truth. Furthermore, the “scaling up” itself might not be an adequate solution for our problem. Suppose HotBot has a very good performance in region A, but does very poorly in region B. Moreover, assume that A and B are included in the population of pages relevant to at least one of the 575 queries. According to the method we employed, we would use the same scaling factor for both regions. If these hypotheses were true, we would obviously reach an erroneous conclusion. Nonetheless, we believe that the situation we described is very unlikely to have actually occurred for the six search engines employed in our study.

## 5 Open Research Questions

1. How to sample from the Web directly, without exploiting the search engines?
2. How to obtain a more reliable estimate of the size of the Web without using a reported size of some search engine index?
3. Are there better ways of expressing/summarizing the amount of information on the Web besides the ones mentioned in this chapter?
4. A new generation of search engines built with different tools, such as Google, has revolutionized Web searches, and the assumptions of the Rasch model are unlikely to hold if we were to look at the engines today. How does that change the way the statistical analyses are performed?

## Acknowledgements

Preparation of this chapter was supported in part by the Center for Automated Learning and Discovery at Carnegie Mellon University under Grant no. REC-9720374 from the National Science Foundation. The authors would like to thank Jennifer Pittman for assistance. Lee Giles and Brian Junker contributed with valuable discussions.

## References

1. R. Albert, H. Jeong, and A. L. Barabasi. Diameter of the World-Wide Web. *Nature*, 401:130, 1999.
2. K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public web search engines. In *Proceedings of the 7th International World Wide Web Conference*, pages 379–388, Brisbane, Australia, April 1998.

3. E. T. Bradlow and D. C. Schmittlein. The little engines that could: Modelling the performance of world wide web search engines. *Marketing Science*, 19:43–62, 2000.
4. A. Dobra and S. E. Fienberg. How large is the World Wide Web? *Computing Science and Statistics - Proceedings of Interface 2001*, 33, 2001.
5. S. E. Fienberg. The multiple recapture census for closed populations and incomplete  $2^k$  contingency tables. *Biometrika*, 59:591–603, 1972.
6. S.E. Fienberg, M. S. Johnson, and B. W. Junker. Classical multi-level and bayesian approaches to population size estimation using multiple lists. *Journal of Royal Statistical Society*, 162:383–406, 1999.
7. M. S. Johnson, W. Cohen, and B. W. Junker. Measuring appropriability in research and development with item response models. Technical Report, Carnegie Mellon University, 1999.
8. S. Lawrence and C. L. Giles. Searching the world wide web. *Science*, 280:98–100, 1998.
9. S. Lawrence and C. L. Giles. Accessibility of information on the web. *Nature*, 400:107–109, 1999.
10. G. Rasch. *Probabilistic Models for Some Intelligence and Attainment Tests*. Niesen and Lydiche, Copenhagen., 1960. expanded 1980 English edition. University of Chicago Press.
11. E. Selberg. *Towards Comprehensive Web Search*. PhD thesis, University of Washington, June 1999. URL: [www.cs.washington.edu/homes/speed/](http://www.cs.washington.edu/homes/speed/).



---

# Methods for Mining Web Communities: Bibliometric, Spectral, and Flow

Gary William Flake<sup>1</sup>, Kostas Tsioutsoulouklis<sup>2</sup>, and Leonid Zhukov<sup>1</sup>

<sup>1</sup> Overture Research, Pasadena, CA 91103, USA

[gary.flake@overture.com](mailto:gary.flake@overture.com), [leonid.zhukov@overture.com](mailto:leonid.zhukov@overture.com)

<sup>2</sup> NEC Laboratories America, Princeton, NJ, 08540, USA

[kt@nec-labs.com](mailto:kt@nec-labs.com)

**Summary.** In this chapter, we examine the problem of Web community identification expressed in terms of the graph or network structure induced by the Web. While the task of community identification is obviously related to the more fundamental problems of graph partitioning and clustering, the basic task is differentiated from other problems by being within the Web domain. This single difference has many implications for how effective methods work, both in theory and in practice. In order of presentation, we will examine bibliometric similarity measures, bipartite community cores, the HITS algorithm, PageRank, and maximum flow-based Web communities. Interestingly, each of these topics relate to one another in a nontrivial manner.

## 1 Introduction

A *Web community* can be loosely defined as a collection of Web pages that are focused on a particular topic or theme. Viewed from the framework of traditional information retrieval, the problem of community identification would be expressed in terms of document content or explicit relations between documents. However, given the hyperlinked structure of the Web, the community identification problem can be reformulated so as to exploit the implicit relations between documents that are formed by hyperlinks.

To leverage the existence of hyperlinks, we model the Web as a graph where vertices are Web pages and hyperlinks are edges. While Web pages may be similar in terms of textual or multimedia content, a hyperlink is usually an explicit indicator that one Web page author believes that another's page is related or relevant. By examining the structure of hyperlinks on the Web, one can identify communities of Web pages that are more tightly coupled to each other than they are to pages outside the community. Using hyperlinks in this manner – independently of or in addition to using text – allows for communities to be identified in a manner that is less sensitive to the subtleties of language. Moreover, by keeping text and hyperlinks separate, one can use the two to covalidate each other.

Mining Web communities is an interesting problem viewed from mathematical, scientific, and engineering viewpoints. Mathematically, the problem is interesting because any effective solution must make a compromise between the quality of the solution and the run-time resources of the algorithm. How these two constraints are traded off touches on deep questions related to fundamental clustering algorithms. Scientifically, identified Web communities can be used to infer relationships between Web communities, the pages that form the communities, and the users that author and visit the community members. Hence, Web community identification can be used to dissect the Web so as to make it a tractable subject of scientific investigation. Finally, from the engineering point of view, Web community identification can be used to build many applications, including portal regeneration, niche search engines, content filters, and personalized search.

So how does community identification differ from mining relations within a database? The Web differs from most databases in the following aspects:

- The Web is completely decentralized in organization and in evolution. Hyperlinks can be strong or weak indicators of relevance, or completely counter-informative in the case of spam; hence, a lack of centralized authorship makes the Web's hyperlinks not only disorganized when viewed on a large scale, but inconsistent when viewed on a small scale (unlike the regularity of relations within a database).
- The Web is enormous [30]. As of 2003, the typical search engine index size is approximately two billion documents, which represents a mere fraction of the whole of the Web. Web pages may not be indexed for a variety of reasons, including prohibition due to robot exclusion rules, falling behind corporate or personal firewalls, being reachable only through a search form, or simply not being popular enough (in in-bound link cardinality) for indexing. Assuming a conservative average document size of 10 KB, the indexable Web is at least 20 TB in size, making it larger than most other databases excepting offline data warehouses.
- The Web is distributed unlike anything else in the world. Not only is the Web distributed over multiple computers, it spans organizations, Internet domains, and continents.

That's the bad news. Worse yet, being related to clustering and partitioning, community identification is NP-complete even with mild assumptions. Superficially, the preceding facts suggest that mining the Web for communities is hopeless. However, there is considerable good news to counterbalance the bad:

- The Web is self-organized and contains a considerable amount of exploitable structure [11]. Hyperlink distributions obey power-law relationships [2]. Many hyperlink patterns (Web rings, hierarchical portals, and hubs and authorities) are found repeatedly on the Web, text and hyperlinks are closely correlated [10], and simple generative network models explain a considerable amount of the Web's structure [32].
- The Web graph is exceedingly sparse. Being read and written (mostly) by humans, typical Web pages have a relatively small number of hyperlinks (only dozens)

when compared to the numbers possible if the Web graph were dense (billions). As a result, the graph structure of the Web can be compactly represented. Moreover, this sparseness makes algorithms much more well-behaved than the worst-case scenario.

- Finally, being structured and sparse, the Web can be successfully mined by approximate techniques that, while technically suboptimal in solution quality, empirically yield very good results.

The bulk of this chapter is devoted to this last point. We begin with a brief introduction to clustering and to bibliographic similarity metrics, which is followed by an introduction to bipartite community cores. Next, we examine the PageRank and HITS algorithms and see how they relate to spectral properties of the Web graph. We follow with a detailed section on maximum flow methods for community identification, which includes theoretical and experimental results. Finally, we conclude with a discussion on future work in this area. Throughout this chapter, we will see how the methods for community identification relate to one another in such a way that each is easier to understand and appreciate when given in the context of the other methods.

## 2 Background

The goal of clustering is to group “similar” objects together while keeping “dissimilar” objects apart, with “similarity” being a function of the attributes of the objects being clustered. More formally, a clustering of  $n$  objects is a partitioning into  $k$  different sets so as to minimize some cost of assigning objects into sets. Regardless of the algorithm and the data source, the quality of a clustering can be measured in multiple ways [8]. No single measure for cluster quality is perfect [28]; instead, all measures for cluster quality actually reflect a trade-off between how mistakes and suboptimalities within a clustering are weighted relative to one another.

The task of community identification can be considered a slight simplification of clustering in that one is only required to identify a grouping of items that are similar to some seed set of elements, denoted  $S$ . In other words, community identification is akin to asking “What are the members of the cluster that contains  $S$ ?” Note that a community identification algorithm can easily be turned into a clustering algorithm (and vice versa), so the distinction between the two is not very significant theoretically. However, in practice, finding a Web community is vastly simpler than finding all Web clusters because of the size of the Web.

It is beyond the scope of this chapter to survey all clustering methods. Instead, we refer the reader to a current survey [8] and general texts [7, 22], but make connections between Web methods and more classical clustering approaches when appropriate.

### 2.1 Notation

Throughout this chapter we will represent the Web (or a subset of the Web) as a directed graph  $G = (V, E)$ , with  $n = |V|$  vertices in set  $V$ , and  $m = |E|$  edges in

set  $E$ . Each vertex corresponds to a Web page, and each directed edge corresponds to a hyperlink. When appropriate, we will refer to an edge by a single symbol, such as  $e \in E$ , or as a node pair  $(u, v) \in E$  with  $u, v \in V$ . In the latter case,  $(u, v)$  is the directed edge that starts at  $u$  and ends at (points to)  $v$ . It may also be necessary to associate a real-valued function with an edge, such as a flow function  $f(e) = f(u, v)$ , or a capacity function  $c(e) = c(u, v)$ .

Graphs can be equivalently represented as an  $n \times n$  adjacency matrix  $\mathbf{A}$  with  $A_{uv} = 1$  if  $(u, v) \in E$  (i.e. page  $u$  links to page  $v$ ) and  $A_{uv} = 0$  if no direct hyperlink exist from  $u$  to  $v$ . Note that all matrices will be written in bold uppercase roman text and vectors in bold lowercase roman text.

The *degree* of a vertex is the number of edges incident to the vertex in question. The *in-degree* (respectively *out-degree*) of a vertex is the number of in-bound (respectively out-bound) edges incident on the vertex. We denote the in-degree and out-degree of  $v$  by  $d_v^{\text{in}}$  and  $d_v^{\text{out}}$ , respectively. Other functions or attributes of vertices will be similarly represented by a symbol with a vertex subscript.

### 3 Bibliographic Metrics and Bipartite Cores

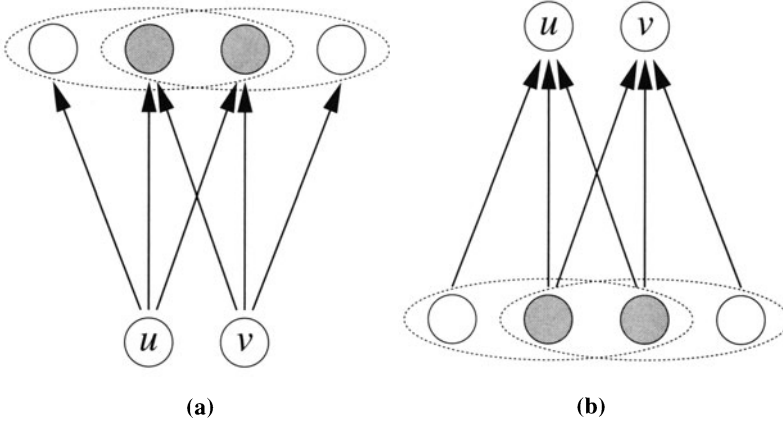
Identification of a Web community can be done in several ways. One of the key distinguishing features of the algorithms we will consider has to do with the degree of locality used for assessing whether or not a page should be considered a community member. On the one extreme are purely local methods, which consider only the properties of the local neighborhood around two vertices to decide if the two are in the same community. Global methods operate at the other extreme, and essentially demand that every edge in a Web graph be considered in order to decide if two vertices are members of the same community. We begin with two related local methods.

#### 3.1 Bibliographic Metrics

We started this chapter by noting that a hyperlink between two pages can be an explicit indicator that two pages are related to one another. However, we should also note that not all related pages are linked. In fact, for many Web pages, what they link to and what links to them may be more informative if considered in the aggregate.

Fig. 1 illustrates two complementary metrics known as *bibliographic coupling* and *cocitation coupling*. In the figure, we see that the two metrics count the raw number of out-bound or in-bound references, respectively, shared by two pages  $u$  and  $v$ . Both metrics were originally formulated to capture similarity between scientific literature [25, 33] by comparing the amount of overlap between the bibliographies or referrers for two different documents [15, 36].

Considering the Web as graph with adjacency matrix  $\mathbf{A}$ , the product  $\mathbf{A}\mathbf{A}^T$  captures the bibliographic coupling between all page pairs such that  $(\mathbf{A}\mathbf{A}^T)_{uv}$  equals the bibliographic coupling between pages  $u$  and  $v$ . Similarly, the product  $(\mathbf{A}^T\mathbf{A})_{uv}$  equals the cocitation coupling between  $u$  and  $v$ .



**Fig. 1.** Graphic portrayal of bibliographic metrics: **(a)** bibliographic coupling. **(b)** cocitation coupling. For vertices  $u$  and  $v$ , the similarity metric is shown as the amount of overlap that vertices have in the set of out-bound neighbors or in-bound neighbors

While bibliographic metrics are simple, intuitive, and elegantly represented by linear algebra, they have several practical shortcomings. First, both matrices  $\mathbf{A}\mathbf{A}^T$  and  $\mathbf{A}^T\mathbf{A}$  may be far more dense than  $\mathbf{A}$ , making storage an issue. Second, some pages may link to or be linked by thousands of other pages, making them have a large amount of link overlap with a disproportionate number of pages.

While the first issue is not easily solved, the second can be easily addressed with normalization. We explain the normalization step in terms of out-bound hyperlinks (i.e. a normalized form of bibliographic coupling), but the basic method can be generalized to hyperlinks in either direction. Let  $U$  and  $V$  denote sets of vertices that are out-bound from pages  $u$  and  $v$ , respectively. The quantities

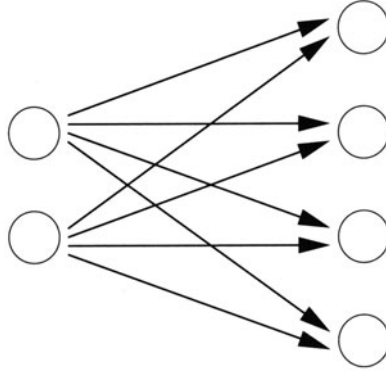
$$\frac{2|U \cap V|}{|U| + |V|} \text{ or } \frac{|U \cap V|}{|U \cup V|} \quad (1)$$

range between 0 and 1 and represent the normalized number of out-bound links shared between  $u$  and  $v$ . One can also generalize normalization methods from text processing, such as  $TF \times IDF$  to work for hyperlinks [17], or even go so far as to use the Pearson correlation between two rows in  $\mathbf{A}$ .<sup>3</sup>

### 3.2 Bipartite Cores

Bibliographic metrics (especially when normalized) are effective for characterizing the degree of similarity between two pages in terms of what they link to and what links to them. What is missing in this framework is the notion that a collection of pages can be related to each other in an aggregate sense.

<sup>3</sup> Make each row (or column) zero mean, unit variance, and take their dot product.



**Fig. 2.** Graphic portrayal of a complete bipartite graph,  $K_{2,4}$ , with each vertex on the *left* (set  $L$ ) linking to each vertex on the *right* (set  $R$ ). As a bipartite core,  $K_{2,4}$  could be embedded within a larger graph that has edges that obscure the bipartite core, for example, linking from vertices in  $R$  to  $L$ ,  $R$  to  $R$ ,  $L$  to  $L$ , or  $V - (L \cup R)$  to  $(L \cup R)$

A *complete bipartite graph* is a directed graph with vertices that can be divided into two sets,  $L$  and  $R$  (for *left* and *right*) with  $L \cup R = V$  and  $L \cap R = \emptyset$ , such that each vertex in  $L$  has an edge to each vertex in  $R$ . We use the notation  $K_{l,r}$  to denote a complete bipartite graph with  $l = |L|$  and  $r = |R|$ . A *bipartite core* is a subgraph that when considered by itself, forms a complete bipartite graph. Note that this definition for a bipartite core does not preclude vertices (in either of  $L$  or  $R$ ) from linking to – or being linked by – other vertices outside of  $L$  and  $R$ , nor does it prohibit edges flowing from  $R$  to  $L$ . The definition states that being fully connected from  $L$  to  $R$  is both necessary and sufficient. Figure 2 shows an example bipartite subgraph,  $K_{2,4}$ .

Bipartite subgraphs are relevant to Web communities for at least two reasons that subtly relate to one another. First, a bipartite core,  $K_{l,r}$ , has the properties that all vertices in  $L$  have a bibliographic coupling value lower-bounded by  $r$  and all vertices in  $R$  have a cocitation coupling value lower-bounded by  $l$ . Thus, bipartite subgraphs consist of vertices that have a minimal degree of similarity in terms of raw bibliographic metrics.

The second reason why bipartite subgraphs are relevant to Web communities is because they empirically appear to be a signature structure of the core of a Web community [29]. Intuitively, authors of Web sites on the same topic may explicitly choose not to link to one another if the sites are competitive with one another (commercial sites in the same industry) or if they are idealistically opposed (pro- $X$  versus anti- $X$  sites). Nevertheless, other Web site authors may choose to link to both competitive sites, thus making the two competitors have a nonzero cocitation coupling. In a similar manner, a reference Web site – say one that contains links to auto companies – will have a link pattern that resembles other auto reference sites. Two general auto references will probably not link to one another, but they will often have a high degree of bibliographic coupling. In this way, the structure of a bipartite core captures the dual nature of Web pages, namely, that general pages link to specific pages, and that

pages about the same general topic or specific topic will often have similar references or referrers.

Kumar et al. [29] used all of these empirical observations to devise an efficient procedure to identify bipartite cores from a Web corpus. In a subset of the Web (approximately 200 million Web pages), Kumar et al. found over 100,000 bipartite community cores, some being as large as  $K_{69}$ . Interestingly, even the smallest identified cores ( $K_{33}$  and  $K_{35}$ ) were topically focused on an identifiable theme in 96% of the sampled examples. Example topics included

- Japanese elementary schools
- hotels in Costa Rica
- Turkish student associations

Hence, the identified community cores were usually topically focused and so specific that they were often not part of any preexisting portal hierarchy. This last point is important because it means that community cores are “natural” in the sense that they are self-organized, and not an artifact of a single individual entity.

## 4 Spectral Methods

The previous section focused on local methods where the affinity between two pages could be determined by examining the local region about one or two pages. In this section we focus on global methods that essentially consider all links in the Web graph (or subgraph) to answer the question “Do these two pages belong with each other?” We begin with a brief survey of linear algebra and eigenvectors. A far better survey of linear algebra can be found in [34].

### 4.1 Linear Algebra and Eigenvectors

Any nondefective  $n \times n$  matrix  $\mathbf{M}$  can be equivalently represented as a summation of vector outer-products:

$$\mathbf{M} = \sum_{i=1}^k \lambda_i \mathbf{r}_i \mathbf{l}_i^T, \quad (2)$$

where  $\mathbf{l}_i$  and  $\mathbf{r}_i$  are respectively the  $i$ th left and right eigenvectors of  $\mathbf{M}$ ,  $\lambda_i$  is the  $i$ th eigenvalue of  $\mathbf{M}$ , and  $\mathbf{M}$  has all of the following properties with respect to its eigenvectors and eigenvalues:

$$\lambda_i \mathbf{r}_i = \mathbf{M} \mathbf{r}_i, \quad (3)$$

$$\lambda_i \mathbf{l}_i = \mathbf{M}^T \mathbf{l}_i, \quad (4)$$

$$\mathbf{l}_i^T \mathbf{l}_i = \mathbf{r}_i^T \mathbf{r}_i = \mathbf{r}_i^T \mathbf{l}_i = 1, \text{ for all } i, \quad (5)$$

$$\mathbf{l}_i^T \mathbf{r}_j = 0, \text{ for } i \neq j, \text{ and} \quad (6)$$

$$\lambda_i \geq \lambda_{i+1} \text{ for all } i. \quad (7)$$

The eigenvalues and eigenvectors form the *spectrum* of a matrix; hence, the reason why algorithms that make use of eigenvectors and eigenvalues are often referred to as *spectral methods*. If the spectrum of a matrix is full (i.e. it contains  $n$  distinct eigenvectors), then either the left or right eigenvectors can be used as a basis to express any  $n$ -dimensional vector. If  $\mathbf{M}$  is symmetric, then the left and right eigenvectors of  $\mathbf{M}$  are identical; however, for an asymmetric matrix, the left and right eigenvectors form a *contravariant basis* with respect to each other, provided that all eigenvectors and eigenvalues are all real.

The key intuition behind the eigen-decomposition of a matrix is that it yields a procedure for compressing a matrix into  $k \leq n$  outer-products, and for expressing matrix–vector products as a summation of inner products. When  $\mathbf{M} = \mathbf{A}^T \mathbf{A}$  for some choice of  $\mathbf{A}$ , then the spectral decomposition of  $\mathbf{M}$  is closely related to the singular value decomposition of  $\mathbf{A}$  [34].

## 4.2 HITS

Kleinberg’s *Hyperlink-Induced Topic Search* (HITS) [26] algorithm takes a subset of the Web graph and generates two weights for each page in the subset. The weights are usually referred to as the *hub* and *authority* score, respectively, and they intimately relate to the spectral properties of the portion of the Web graph for which the algorithm is being used.

Conceptually, a *hub* is a Web page that links to many authorities, and an *authority* is a Web page that is linked by many hubs. The two scores for each page characterize to what degree a page obeys the respective property. Referring back to Fig. 2, we can see that pages on the left side of a dense bipartite core are hubs, and pages on the right are authorities.

HITS is actually performed in two parts. The first is a preprocessing step used to select the subset of the Web graph to be used, while the second part is an iterative numerical procedure. The first part usually proceeds as follows:

1. Send a query of interest to the search engine of your choice.
2. Take the top 200 or so results from the search engine.
3. Also identify all Web pages that are one or two links away (in either direction) from the results gathered in the previous step.

All told, the first part generates a *base set* of Web pages that either contain the original query of interest, or are within two links away from a page that does. Of course, other heuristic constraints need to be used, such as limiting the total number of pages, only considering interdomain hyperlinks, and changing the size of the initial result set and/or the diameter of the base set.

With the base set of pages being generated, let  $G = (V, E)$  refer to this subset of the Web graph (with intradomain hyperlinks removed)<sup>4</sup> and let  $\mathbf{A}$  be this graph’s adjacency matrix.

<sup>4</sup> This step reduces the impact of nepotism.



The iterative numerical part of HITS updates two  $|V| \times 1$  dimensional vectors,  $\mathbf{h}$  and  $\mathbf{a}$ , as follows, with the initial values of both vectors being set to unity:

$$\mathbf{a} = \mathbf{A}^T \mathbf{h}, \quad (8)$$

$$\mathbf{h} = \mathbf{A} \mathbf{a}, \quad (9)$$

$$\mathbf{a} = \mathbf{a} / \|\mathbf{a}\|^2, \quad (10)$$

$$\mathbf{h} = \mathbf{h} / \|\mathbf{h}\|^2. \quad (11)$$

In plain English, Eq. 8 says “let a page’s authority score be equal to the sum of the hub scores of the pages that link to it,” while Eq. 9 says “let a page’s hub score be equal to the sum of the authority scores that it links to.” The remaining equations enforce  $\mathbf{h}$  and  $\mathbf{a}$  to maintain unit length.

After iterating the equations, we select the authority pages to be those with the largest corresponding value in  $\mathbf{a}$ , and the hub pages to be the ones with the largest corresponding value in  $\mathbf{h}$ .

HITS is a close cousin to the *power method* [34] for calculating the eigenvector of a matrix with the largest eigenvalue (the maximal eigenvector). Both procedures converge to a solution in  $k$  iterations with error proportional to  $O(|\lambda_2/\lambda_1|^k)$ . Hence, the procedure can be slow, but it tends to be fast for power law graphs which often have the property that  $\lambda_1 \gg \lambda_2$  [5]. With minimal substitution, we can see that  $\mathbf{h}$  and  $\mathbf{a}$  converge to the maximal eigenvectors of  $\mathbf{A}\mathbf{A}^T$  and  $\mathbf{A}^T\mathbf{A}$ . Thus, HITS produces a rank one approximation to the raw bibliographic and cocitation coupling matrices.

### 4.3 PageRank

The PageRank algorithm [3] is motivated by a random walk model of the Web. At any given time step, a random walker exists on a Web page and can do one of two things: (1) with probability  $\epsilon$  it can “teleport” to a random Web page (chosen uniformly from all pages), or (2) with probability  $(1 - \epsilon)$  it can move from its current location to a random page that it currently points to. If we repeat these steps, each Web page will ultimately have a stable and computable probability of being visited by the random walker; this probability is equivalent to the PageRank value of a page.

We can explicitly calculate the PageRank  $r_v$  of page  $v$  with the following iterative procedure:

$$r_v^{t+1} = \frac{\epsilon}{n} + (1 - \epsilon) \sum_{u:(u,v) \in E} \frac{r_u^t}{d_u^{\text{out}}}, \quad (12)$$

where  $\epsilon$  is typically set to something small (say 0.1), and superscripts on  $r_v$  are used only to indicate time dependencies and that all values should be simultaneously updated. Additionally, after each iteration,  $\mathbf{r}$  must be normalized<sup>5</sup> so that  $\sum r_v = 1$ .

Intuitively, PageRank enforces the recursive idea that pages are important if important pages link to them. PageRank is used as a means to effectively boost the ranking of important Web pages that all match the same query.

<sup>5</sup> The normalization step is required only when  $G$  is not ergodic.

PageRank relates to the spectral properties of  $\mathbf{A}$  in the following way. Let  $\mathbf{M}$  be equal to  $\mathbf{A}$  except that  $\mathbf{M}$  has its rows normalized so that they all sum to 1. Let  $\mathbf{U}$  be the  $n \times n$  matrix with all components equal to  $\frac{1}{n}$ . The PageRank vector  $\mathbf{r}$  is the maximal eigenvector of

$$(\epsilon \mathbf{U} + (1 - \epsilon) \mathbf{M})^T, \quad (13)$$

provided that  $G$  is ergodic, which means that in the limit (as time goes to infinity), the random walker has a nonzero probability of revisiting every page. A Web graph with pages that have no out-bound links or sink regions – regions that once entered cannot be escaped through forward links – break this constraint. Nonetheless, it is helpful to see that PageRank converges to the maximal eigenvector of something like a “well-behaved” version of  $\mathbf{A}$  (where “well-behaved” means ergodic and out-bound weight normalized).

#### 4.4 Discussion

Interestingly, HITS has a random walk interpretation similar to that of PageRank [31]. A HITS random walker does not need the “teleport” step, but it does need to alternate between two different types of moves done relative to its current location: (1) follow a random out-bound link, (2) follow a random in-bound link (i.e. as if to reverse the direction). If these moves are repeatedly done in order (move (1) followed by move (2)), then after each pair of moves, the random walker will be at a page that has some nonzero amount of bibliographic coupling with the page that it was at the previous step. The greater the bibliographic coupling, the greater the probability that the random walker will end up at a page. If we were to run this procedure forever, then the probability of visiting a page would be proportional to the hub score of a page (with scaling differences due to normalizing the sum to 1 instead of the sum of squares to 1).

If we reorder the moves and do (2) followed by (1), then the random walker will have behavior that is characterized by the authority score vector. In this scenario, the probability of moving from one page to another is related to the amount of raw cocitation coupling between two pages.

In either case, it is interesting to see that both HITS and PageRank can be described in terms of the spectral properties of an adjacency matrix as well as the long-term behavior of a random walker. However, at first glance, neither HITS nor PageRank appear to be methods for community identification. HITS uses an initial query to limit the neighborhood of the Web graph from which to score pages, and PageRank is usually coupled to a secondary text retrieval step that is used to filter results. Hence, in the usual implementation, each method has a crucial dependency on text. However, both HITS and PageRank can be adapted to the problem of community identification.

#### HITS Communities

Recall from Eq. 2, that a matrix can be rewritten as a summation of outer products. Because both of the matrix products  $\mathbf{A}^T \mathbf{A}$  and  $\mathbf{A} \mathbf{A}^T$  are symmetric and positive

definite, each will have the property that the left and right eigenvectors will be identical (because of symmetry) and that the first eigenvector will have all positive components (with positive eigenvalue).

All other eigenvectors for these matrices can be heterogeneous in that their elements can have mixed signs. These subsequent eigenvectors can be used to separate pages into different communities in a manner related to more classical spectral graph partitioning [4], or in a manner that is related to principal component analysis [23].

Kleinberg [27] and his collaborators [16] have found that the nonmaximal eigenvectors can be used to split pages from a base set into multiple communities that contain similar text but are dissimilar in meaning. Examples include:

- *Abortion*: with communities split along pro-choice, and pro-life camps
- *Jaguar*: with communities split among the various meanings (football team, animal, car, operating system, video game, etc.)

In this manner HITS can be adapted for community identification. The main caveat to spectral approaches is that as the sizes of the communities get smaller, the less significant eigenvectors can be dominated by noise and confused by paths of longer length. Nevertheless, the approach has considerable power, and spectral methods offer a very elegant mathematical derivation.

### PageRank Communities

PageRank can be generalized into a form known as *topic-sensitive PageRank* [21], which replaces the uniform “teleportation” term in Eq. 12 with a nonuniform term

$$r_v^{t+1} = \epsilon p_v + (1 - \epsilon) \sum_{u:(u,v) \in E} \frac{r_u^t}{d_u^{\text{out}}}, \quad (14)$$

with the constraint that  $\sum_v p_v = 1$ . The value of  $p_v$  (for all  $v$ ) is chosen so that it reflects the probability that the random walker should land on a page  $v$  during one of its “teleportation” steps.

The intuition behind topic-sensitive PageRank is that the random walker moves about as before with the exception that on “teleportation” the walker moves to a page that is focused on some particular topic. For all known pages,  $v$ , on the topic, we can set  $p_v$  to some nonzero value. In this way, the random walker is similar to a Web surfer that browses as normal, but periodically restarts to a smaller set of favorite bookmarked pages.

This basic procedure can be used to identify a community of pages by using the topic -sensitive score to filter pages. Pages that belong to the community are those that have a probability of being visited greater than some threshold; everything else is outside the community.

## 5 Maximum Flow Communities

So far, we have characterized community identification methods as being either local or global. In this section we will consider the *community algorithm* [9, 10], which

has both local and global properties. It can operate over the entire Web graph or a subgraph; it has worst-case time complexity that is a function of the whole of the Web, yet in practice it is fast because its run time is often dominated by the size of the community that it finds (and not the whole graph). Moreover, it yields communities that have strong theoretical guarantees on their local and global properties.

The community algorithm achieves these properties by recasting the problem into a maximum flow framework. We begin with a subsection describing  $s$ - $t$  maximum flows, minimum cuts, and a simple procedure for solving the  $s$ - $t$  maximum flow problem. Next, we describe the community algorithm along with a few variations of it. We follow with analysis, less formal discussion, and conclude this section with experimental results.

### 5.1 Max Flows and Min Cuts

While flows and cuts are well-defined for both directed and undirected graphs, we will restrict the domain to undirected graphs to simplify some definitions and to clarify later analysis. Note that any directed graph can be converted into an undirected graph by treating each edge as being undirected. Thus, let  $G = (V, E)$  be henceforth understood as an undirected graph, and for all edges  $(u, v) \in E$  let  $c(u, v)$  denote the capacity of an edge. By convention, we say that  $c(u, v) = 0$  if  $(u, v)$  does not exist.

Given two vertices,  $s$  and  $t$ , the  $s$ - $t$  maximum flow problem is to find the maximum flow that can be routed from  $s$  to  $t$  while obeying all capacity constraints of  $c(\cdot)$  with respect to  $G$ . Intuitively, if edges are water pipes and vertices are pipe junctions, then the maximum flow problem tells you how much water you can move from one point to another.

Ford and Fulkerson's [14] "max flow-min cut" theorem proves that the  $s$ - $t$  maximum flow of a graph is identical to the minimum cut that separates  $s$  and  $t$ . The intuition behind the theorem is that flows are limited by bottlenecks, and by removing these bottlenecks, one can separate two points in a network. Many polynomial time algorithms exist for solving the  $s$ - $t$  maximum flow problem and the curious reader should definitely consult a dedicated text [1] for a more thorough discussion of cuts and flows, or the excellent survey found in [18]. In any event, we describe the simplest flow algorithm to better convey some intuition behind the problem and the algorithm.

Table 1 gives pseudocode for the augmenting path maximum flow algorithm, which is the simplest maximum flow algorithm known. The procedure operates on a *residual network* which is a data structure used to keep track of edge capacities, both used and available. The residual network  $R = (V, E')$  of  $G$  has two directed edges for every undirected edge in  $E$ ; hence, for  $(u, v) \in E$ ,  $E'$  will have both  $(u, v)$  and  $(v, u)$ . The residual capacities in  $R$  are initialized by  $r(u, v) = r(v, u) = c(u, v)$  for all  $(u, v) \in E$ .

We say that  $R$  has an *augmenting path* from  $s$  to  $t$  if there exists a path connecting the two vertices such that each directed edge along the path has nonzero residual capacity. At line 5 of the procedure, we identify the smallest capacity value along the

**Table 1.** The augmenting path maximum flow algorithm.

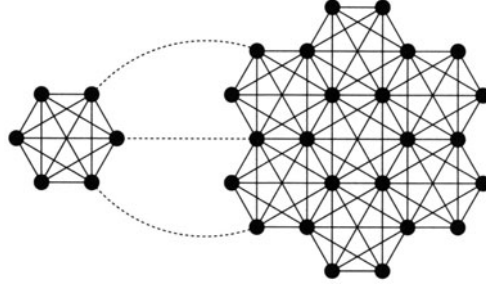
---

```

procedure MAX-FLOW(graph:  $G = (V, E)$ ; vertex:  $s, t$ )
  set  $R \leftarrow$  residual network of  $G$ 
  while  $R$  contains a directed path from  $s$  to  $t$  do
    identify shortest augmenting path,  $P$ , from  $s$  to  $t$ 
    set  $\delta = \min(r(u, v) : (u, v) \in P)$ 
    for all  $(u, v) \in P$  do
      set  $r(u, v) \leftarrow r(u, v) - \delta$ 
      set  $r(v, u) \leftarrow r(v, u) + \delta$ 
    end for
  end while
  return  $R$ 
end procedure

```

---

**Fig. 3.** Maximum flow methods will separate the two subgraphs with any choice of  $s$  and  $t$  that has  $s$  on the *left subgraph* and  $t$  on the *right subgraph*, removing the three dashed links

path  $P$ . Lines 6–9 remove the available capacity from the residual network along the path; if  $r(u, v)$  becomes zero, we treat that edge as no longer being available. In this way, the procedure simply forces flow from  $s$  to  $t$  until no more flow can be passed. Finally, at line 11, when there are no more paths from  $s$  to  $t$ , we return  $R$ , which contains sufficient information to easily find the  $s$ - $t$  minimum cut or maximum flow of  $G$ .  $R$  can also be used to find a connected component that contains  $s$ , which will be important for the algorithms that follow.

## 5.2 The Community Algorithm

We now define communities in terms of an undirected graph where each edge has unit capacity. The definition can be generalized for non-unit capacity edges, but cannot be easily generalized for the directed case.

**Definition 1.** A COMMUNITY is a vertex subset  $X \subseteq V$ , such that for all vertices  $v \in X$ ,  $v$  has at least as many edges connecting to vertices in  $X$  as it does to vertices in  $(V - X)$ .

Note that this definition is slightly recursive in that it leads to statements of the form “a Pokémon Web site predominately links more Pokémon sites than non-Pokémon sites.” Figure 3 shows an example of a community (on the left) being separate from the rest of the graph (on the right).

Interestingly, the question “Does some graph contain a nontrivial community?” is NP-complete [35]. However, we will show that there is a polynomial time algorithm that can identify many communities (but not all). This is not as much of a sacrifice as it would seem because the procedure’s limitation is that it can only find communities that have a high degree of intracommunity weight and a low degree of intercommunity weight, that is, the communities that it is limited to finding are in fact “strong” communities.

The COMMUNITY procedure is shown in Table 2. Its input is a graph  $G$ , a set of “seed” Web sites  $S$ , and a single parameter  $\alpha$ , which will be explained in greater detail in Sect. 5.3. The procedure creates a new graph,  $G_\alpha$  that has two artificial vertices  $s$  and  $t$ . The source vertex  $s$  is connected with infinite capacity to all pages in the seed set,  $S$ . The sink vertex,  $t$  is connected to all original vertices with a small capacity specified by  $\alpha$ .

After constructing  $G_\alpha$ , the procedure calls MAX-FLOW as a subroutine, and uses the resulting residual graph to return the portion of  $R$  that remains connected to  $s$ . This connected component is guaranteed to be a community as defined by Definition 1, provided that the algorithm has not terminated with the trivial cut of simply disconnecting all  $v$  in  $S$  from the rest of the graph.

Tables 3 and 4 contain two related algorithms, APPROXIMATE-COMMUNITY and COMMUNITY-CLUSTER, respectively (the latter referred to as “cut clustering” in [12, 13]), both of which use COMMUNITY as a subroutine. Procedure COMMUNITY is appropriate when the entire graph can be contained in memory and only a single community is required. Procedure APPROXIMATE-COMMUNITY is appropriate when only a small portion of the graph can be contained in memory. It uses a fixed depth crawl to calculate an approximate community, then uses the “strongest” members in the community to be new seeds for a subsequent iteration. Procedure COMMUNITY-CLUSTER can be used to find all communities in a graph. Hence, the last procedure is only appropriate when all communities are desired and the entire graph can be maintained in memory.

### 5.3 Analysis

Having formally defined the community algorithm, we can now make some rigorous statements regarding the quality of maximum flow communities relative to inter- and intracommunity weight. But first, some definitions.

Let the maximum flow value between  $s$  and  $t$  be represented as  $f(s, t)$ . We denote the edge cut set that separates  $s$  and  $t$  with total weight  $f(s, t)$  by  $C(s, t) \subseteq E$ .

**Table 2.** The community algorithm.

---

```

procedure COMMUNITY(graph:  $G = (V, E)$ ; set:  $S$ ; real:  $\alpha$ )
  set  $V_\alpha \leftarrow V \cup \{s, t\}$ 
  set  $E_\alpha \leftarrow E \cup \{(v, t) : v \in V\} \cup \{(s, u) : u \in S\}$ 
  set  $c(v, t) \leftarrow \alpha, \forall v \in V$ 
  set  $c(s, u) \leftarrow \infty, \forall u \in S$ 
  set  $G_\alpha \leftarrow (V_\alpha, E_\alpha)$ 
  set  $R \leftarrow \text{MAX-FLOW}(G_\alpha, s, t)$ 
  set  $X \leftarrow \text{members of smallest connected component about } s \text{ in } R$ 
  return  $X - \{s\}$ 
end procedure

```

---

**Table 3.** The approximate community algorithm.

---

```

procedure APPROXIMATE-COMMUNITY(set :  $S$ ; integer:  $k$ )
  while number of iterations is less than desired do
    set  $G$  to a crawl from  $S$  of depth  $k$ 
    set  $\alpha \leftarrow |S|$ 
    set  $X \leftarrow \text{COMMUNITY}(G, S, \alpha)$ 
    rank all  $v \in X$  by number of edges in  $X$ 
    add highest ranked non-seed vertices in  $X$  to  $S$ 
  end while
  return  $X$ 
end procedure

```

---

**Table 4.** The community clustering (or cut clustering) algorithm.

---

```

procedure COMMUNITY-CLUSTER(graph:  $G = (V, E)$ ; real:  $\alpha$ )
  set  $S \leftarrow V$ 
  while  $\exists s \in S$  do
    set  $X \leftarrow \text{COMMUNITY}(G, \{s\}, \alpha)$ 
    for all  $v \in X$  do
      set  $\text{cluster}(v) \leftarrow s$ 
    end for
    set  $S \leftarrow S - X$ 
  end while
  return cluster labels of  $v \in V$ 
end procedure

```

---

Removing the cut set  $C(s, t)$  from  $E$  will always leave at least two connected components: one that contains  $s$  and the other that contains  $t$ . The maximum flow always has the following relationship to the cut set:

$$f(s, t) = \sum_{(u,v) \in C(s,t)} c(u, v). \quad (15)$$

Finally, we can generalize the meaning of  $C(\cdot)$  and  $f(\cdot)$  so that their arguments range over sets of vertices. In this case,  $C(S, T)$  will be the edge cut set of minimal capacity that separates all vertices in  $S$  from all vertices in  $T$ , and  $f(S, T)$  is the maximum flow or minimum cut value between the two sets.

Recall that the `COMMUNITY` procedure uses a single parameter  $\alpha$ , which serves as the capacity between all vertices and the artificial vertex  $t$  added to  $G$ . The main theoretical result for the community algorithm follows and is made with reference to Fig. 4.

**Theorem 1.** *Let  $X$  be a community found by procedure `COMMUNITY` with value  $\alpha$ . For any  $P$  and  $Q$  such that  $P \cup Q = X$  and  $P \cap Q = \emptyset$ , the following bounds will always hold:*

$$\frac{f(X, V - X)}{|V - X|} \leq \alpha \leq \frac{f(P, Q)}{\min(|P|, |Q|)}.$$

Proof of the theorem is beyond the scope of this chapter, but the complete proof can be found in [12], [35], or [13].

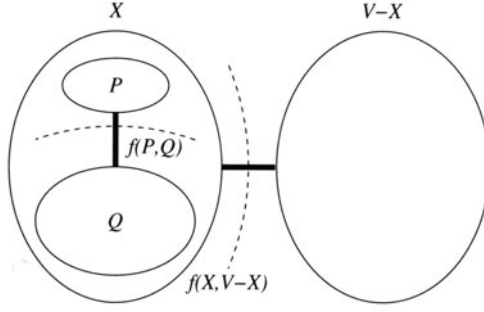
In a nutshell, Theorem 1 shows that  $\alpha$  serves as an upper bound for the inter-community edge capacity, and a lower bound for the intracommunity edge capacity. Thus, the community algorithm simultaneously guarantees that community members are relatively densely connected to one another but relatively sparsely connected to non-community members.

Also notice that these bounds show how  $\alpha$  can be used to tune the size and number of identified communities. A small choice for  $\alpha$ , say close to zero, can yield just one community that comprises the entire graph. A large value for  $\alpha$ , say  $\alpha = 1 + \sum_{(u,v) \in E} c(u, v)$ , will yield  $n$  singleton communities.

Ironically, the strength of the bounds of the Community Algorithm is in some sense its main weakness. It is possible that there exists a community that obeys Definition 1 but cannot be found because it fails to obey the bounds in Theorem 1. Nevertheless, this price basically means that the algorithm will only find the best communities, where “best” is in terms of the bounds.

Looking back to Table 4, where the `COMMUNITY-CLUSTER` procedure is defined, we can make a few more interesting statements about how communities relate to one another. For any two identified communities,  $X_u$  and  $X_v$ , found by using a seed set of one vertex in each case ( $u$  and  $v$ ), either  $X_u \cap X_v = \emptyset$ ,  $X_u \subseteq X_v$ , or  $X_v \subseteq X_u$  must be true. In other words, communities must nest with one another, forming a hierarchy. This nesting property has been used as a heuristic to speed up the `COMMUNITY-CLUSTER` [12, 13].





**Fig. 4.** Intercommunity and intracommunity cut bounds:  $X \subset V$  is any community and  $P \cup Q = X$  is any possible partitioning of  $X$ .  $f(P, Q)$  is an intracommunity cut value, and  $f(X, V - X)$  serves as an upper bound on intercommunity cuts

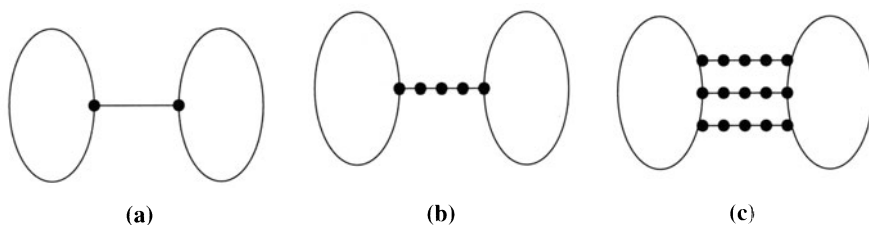
## 5.4 Discussion

The real trick behind the `COMMUNITY` procedure is the transformation of  $G$  that is performed in lines 2–6 in Table 2. The new graph  $G_\alpha$  has flow and cut properties that are similar to  $G$ . However, the artificial sink  $t$  is nearly identical to the centroid of  $G$ 's minimum cut tree (a cut tree is a data structure that preserves all cut and flow properties of a graph; see [19] for more details). This means that using  $t$  as a sink for an  $s$ - $t$  maximum flow calculation is similar to extracting the subtree of  $G$ 's cut tree that contains  $s$ .

Recall that the “teleportation” step in PageRank is required to keep the dynamics of the iterative system well behaved; without it, excess rank could accumulate to pages that have no out-bound links. Almost the exact same correction could be achieved by introducing an artificial vertex that is connected to every other vertex with some small weight or capacity (and in both directions). Thus, a random walker at any vertex would have some small probability of jumping to the artificial vertex. Similarly, a random walker at the artificial vertex could jump to any other vertex with equal probability. In this way, PageRank is similar to the flow-based community algorithm in that both use artificial transitions (via “teleportations” or the artificial edges) to stabilize a calculation.

A significant difference between the flow and spectral methods is in how they treat distances in graphs. HITS and PageRank are sensitive to the distance of the paths that join different parts of a graph. The flow-based community algorithm has no sensitivity to path distance whatsoever. For both spectral methods, a page is a community member if the probability of a random walker visiting the page is above some threshold. For the flow method, a page is a community member if it has more unique paths to community members than to non-community members.

These differences are best visualized with the help of Fig. 5. To a flow algorithm, Fig. 5(a) and 5(b) look identical because the two halves can be disconnected by removing one edge (and the flow is also the same). But to a random walker, Fig. 5(a)



**Fig. 5.** Three similar graphs with different cut and spectral properties: (a) two halves separated by a single short path, (b) two halves separated by a single long path and (c) two halves separated by multiples long paths

and 5(b) look very different because moving between the two halves is much harder in Fig. 5(b) because a long path must be traversed.

By way of comparison, Fig. 5(a) and 5(c) look very different to a flow algorithm because both the cut and the flow between the two halves are very different. Yet to a random walker, Fig. 5(a) and 5(c) are similar because the redundant paths in Fig. 5(c) make up for the fact that the individual paths are long.

The spectral and flow methods also differ in that the spectral methods apply a threshold to page scores after the main portion of the algorithm to decide community membership, while the community algorithm has its equivalent parameter accounted for at the start of the algorithm. This difference may have implications for both run time and quality of results.

In the end, each approach has implicit assumptions built in that make each more practical than many classical clustering or partitioning algorithms; however, the implicit assumptions have implications for how and when each approach makes a mistake.

## 5.5 Experimental Results

To illustrate how the flow-based community algorithm works in practice, we used APPROXIMATE-COMMUNITY to identify a set of Web sites dealing with the topic of September 11, 2001 (i.e. the 9/11 community). Ten Web pages were used as seed sites, and the algorithm was run for four iterations. After completion, the community consisted of 6257 Web sites.

Tables 5, 6, and 7 show three different samples of the community. Table 5 shows pages from the site Howstuffworks. Table 6 shows all pages that have the word “who” in the title. Table 7 is a list of all pages from CNN.com published in 2002.

While it is impossible to enumerate all members of the 9/11 Web community, it is interesting to see how highly relevant the members appear to be based on these samples. Had we simply enumerated pages that contain the words “Bin Laden”, there would be no surprise that some results seem relevant. However, these three tables indicate how the content varies when looking at specific sites, extremely general words, and relatively new additions to the community.

**Table 5.** Members of the 9/11 community that contain the word “Howstuffworks”

- 
- Howstuffworks “How Airport Security Works”
  - Howstuffworks “How Biological and Chemical Warfare Works”
  - Howstuffworks “How Black Boxes Work”
  - Howstuffworks “How Building Implosions Work”
  - Howstuffworks “How Cell Phones Work”
  - Howstuffworks “How Cipro Works”
  - Howstuffworks “How Cruise Missiles Work”
  - Howstuffworks “How Emergency Rooms Work”
  - Howstuffworks “How Machine Guns Work”
  - Howstuffworks “How NATO Works”
  - Howstuffworks “How Nostradamus Works”
  - Howstuffworks “How Nuclear Bombs Work”
  - Howstuffworks “How Skyscrapers Work”
  - Howstuffworks “How Stun Guns Work”
  - Howstuffworks “How the U.S. Draft Works”
  - Howstuffworks “How Viruses Work”
- 

**Table 6.** Members of the 9/11 community that contain the word “who”

- 
- BBC News — SOUTH ASIA — Analysis: Who are the Taleban?
  - BBC News — SOUTH ASIA — Who are the Taleban?
  - BBC News — SOUTH ASIA — Who is Osama Bin Laden?
  - CNN.com - Backgrounder: Who is bin Laden, al Qaeda? - December 12, 2001
  - Countries Need To Plan Effectively for “Deliberate Infections” - WHO Leader Urges Health Ministers
  - DefenseLINK News: Bush: No Distinction Between Attackers and Those Who Harbor Them
  - EIRC is working on a list of Resources to assist school staff respond to the needs of students and their families who may be impacted
  - FEMA: Message to All Who Want to Volunteer or Make Donations from FEMA Director Joe M. Allbaugh
  - Forbes.com: Who Is Osama Bin Laden?
  - frontline: hunting bin laden: who is bin laden?: a biography of osama bin laden
  - Guardian Unlimited — The Guardian — Man whose job is to keep America safe
  - Scoop: David Miller: Who is Osama bin Laden?
  - THE WORLD TRADE CENTER BOMB: Who is Ramzi Yousef? And Why It Matters - The National Interest, Winter, 1995/96
  - U.S. to Assist Those Who Seek a Peaceful, Economically Developed Afghanistan
  - Who did it? Foreign Report presents an alternative view
  - Who is Osama bin Ladin?
  - Who's Clueless?
  - Who's OK? - WorldTradeAftermath.com
- 

**Table 7.** Members of the 9/11 community from CNN.com published in 2002

- 
- CNN.com - Daniel Pearl, 38, reporter, expectant father - February 22, 2002
  - CNN.com - Detainees treated humanely, officials say - January 23, 2002
  - CNN.com - Guantanamo Bay in U.S. control over 100 years - January 10, 2002
  - CNN.com - Police: Tampa pilot voiced support for bin Laden - January 7, 2002
  - CNN.com - States eye high-tech drivers' licenses - February 17, 2002
  - CNN.com - Student Bureau: ‘Suspicion’ - January 21, 2002
  - CNN.com - U.S. journalist Daniel Pearl is dead, officials confirm - February 22, 2002
  - CNN.com - Will anonymous e-mail become a casualty of war? - February 13, 2002
-

**Table 8.** The top 150 text features of the 9/11 Web community. *Underscores* are used to indicate the occurrence of white space between individual words. A *prefix* of A, F, or E indicates that the term occurred in the anchor text, full text, or extended anchor text (anchor text including some extra surrounding text) of a page

1-50	51-100	101-150	151-200
A.terrorism	F.al.qaeda	F.pakistan	A.security
F.terrorist.attacks	A.fbi	A.mil	F.on.september
F.bin.laden	A.wtc	F.the.september	F.u.s.department
E.terrorism	A.laden	A.september	F.crisis
F.taliban	A.attack	F.national.security	F.emergency
F.on.terrorism	F.attacks.on	F.bbc	F.americans
F.in.afghanistan	A.terrorism.http.www	F.arab	A.2001
F.osama	F.afghanistan.and	F.of.war	F.11th
F.terrorism.and	F.attack.on.america	F.state.department	F.the.middle.east
F.osama.bin	F.of.terrorist	F.victims.of	F.troops
F.terrorism	A.bin.laden	F.iraq	F.9.11
F.osama.bin.laden	F.trade.center	F.briefing	E.war
F.terrorist	A.afghan	F.victims	F.threat
E.afghanistan	A.afghanistan	F.tragedy	F.cia
F.against.terrorism	F.of.september.11	A.military	F.the.war
F.the.taliban	F.world.trade	F.attack	A.government
E.terrorist	F.the.pentagon	A.war	F.of.u.s
F.to.terrorism	F.war.against	F.events.of	F.s.department.of
A.state.gov	F.war.on	F.cnn	F.united.nations
F.anthrax	A.defense	F.white.house	F.in.new.york
F.terrorist.attack	F.september.11	F.warfare	A.law
F.world.trade.center	F.president.bush	A.politics	F.center.and
F.the.attack	E.september.11	F.iran	F.enforcement
F.terrorists	A.bbc.co.uk	F.of.defense	A.pubs
A.terrorist	A.bbc.co	A.against	A.federal
E.attacks	F.pentagon	F.department.of.defense	F.2001.the
F.afghan	A.bbc	F.bush	F.defense
F.laden	F.september.11.2001	F.saudi	A.united
F.war.on.terrorism	F.attack.on	F.armed	F.military
A.terror	F.bombing	F.disaster	A.radio
F.afghanistan	F.attacks	F.weapons	F.secretary.of
F.homeland	F.aftermath	F.u.s.government	F.s.department
F.the.world.trade	A.www.state	F.s.government	F.of.international
F.on.america	A.http.www.state	F.law.enforcement	F.justice
F.the.terrorist.attacks	F.department.of.state	F.destruction	A.new.york
A.terrorism.http	F.11.2001	F.the.events	F.of.u
F.the.terrorist	F.islam	F.middle.east	F.congressional
A.emergency	F.red.cross	F.of.state	F.crime
F.of.afghanistan	A.response	F.threats	F.relief
F.the.attacks	F.muslim	F.human.rights	F.killed
F.the.september.11	A.http.news	A.u.s	A.archives
F.september.11th	F.islamic	A.middle	F.coalition
F.wtc	F.muslims	A.asia	A.trade
F.of.terrorism	F.fbi	F.civilian	F.york.times
A.attacks	E.attack	F.speeches	F.new.york.times
A.www.state.gov	F.attack.on.the	F.washington.post	F.nations
F.sept.11	F.september.2001	F.violence	E.september
T.terrorism	F.terror	F.americas	F.the.nation
F.attacks.on.the	F.foreign.policy	A.america	F.and.security
F.qaeda	F.of.september	F.sept	F.nation

As can be seen, the Howstuffworks.com results are very relevant to 9/11 but in a very subtle and nontrivial manner. The pages with “who” in the title appear to have a single outlier (“Who’s Clueless”), yet this Web site was topically focused on the events of 9/11 at the time that this experiment was conducted. Finally, the CNN.com pages show that newer pages in the community maintained relevance to the central theme.

Finally, Table 8 shows the top  $n$ -grams found in members of the 9/11 Web community, ranked by their ability to disambiguate community members from non-community members. (In this case, we used the simplistic ratio of the probability of occurring in the 9/11 community versus the probability of appearing in a random set of pages.) As can be clearly seen, the  $n$ -grams in the table are all highly related to 9/11 with no obvious outliers. These results support the main claim that one can identify large Web communities that are topically and textually focused with methods that use hyperlinks and no text.

More information on the 9/11 community is available at:

<http://webselforganization.com/example.html>

which allows one to browse the entire community and to search the community for specific words.

## 6 Conclusions

We have explored several different methods for identifying communities on the Web. While all of these methods differ from one another, there are interesting connections between many of the methods. For example,

- Bibliometric methods define a notion of similarity for pages that do not directly link to one another.
- Bipartite cores consist of pages that have high bibliographic metrics with respect to each other.
- HITS identifies hubs and authorities, which are pages that often fulfill the definition of a bibliographic core.
- PageRank and HITS both have a spectral and random walk interpretation relative the Web graph.
- PageRank and the flow-based community algorithm, both use a nearly identical artificial transition for stabilizing each respective calculation.

We also saw how some of the approaches related to more traditional clustering algorithms and how in some cases tight theoretical bounds could be derived for global and local community properties.

## 7 Further Explorations

There are many open areas within the field of Web data mining that are directly related to the task of identifying Web communities

## 7.1 Combine Text with Hyperlinks

Probabilistic versions of HITS have been used [6] that treat hyperlinks and text on equal footing. There is an opportunity to do the same for the flow-based community algorithm. In particular, it would be interesting to see if text and hyperlinks could be combined in such a way that accounts for varying word frequencies.

## 7.2 Unite PageRank and the Community Algorithm

The Community Algorithm normally operates over unit capacity edges. However, there has been some interesting success in normalizing edge capacity by out-degree (before discarding edge direction) [35]. An extension of this idea would use PageRank to calculate a probability of an edge transition, then use rescaled probabilities for capacity values. It would also be interesting if this procedure yields more satisfactory bounds on community quality because it may be possible to express the bounds in terms of an absolute probability of moving from one community to another (instead of flow or cut values).

## 7.3 Flow Communities of Bibliographic Matrices

One may also be able to apply the community algorithm to  $A^T A$ ,  $AA^T$ , or some other related matrices. It is not clear when or if this would be a benefit, nor what the precise theoretical interpretation of the resulting communities would be. Nonetheless, these communities may be more appropriate for certain graph families.

## 7.4 Generalize the Community Algorithm for Arbitrary Cohesiveness

It has been suggested [9] that one can generalize the community algorithm so that it finds communities that obey the definition that members have  $x\%$  of their links to other members, instead of the standard 50%. Generalizing the algorithm in this manner would require more extensive use of artificial vertices and edges; however, it may yield more satisfying results and more interesting bounds.

## 7.5 Use Randomized Algorithms for Improved Speed

While flow algorithms have been steadily improving, there exist approximate and randomized approaches [24] applicable to cut and flow problems that may improve performance on large-scale problems. In particular, it would be interesting to implement such an algorithm that could operate on a graph with only linear scans of the adjacency matrix.

## 7.6 Automate the Search for $\alpha$

Because the community algorithm's  $\alpha$  parameter reflects a trade-off between community size and the number of communities in a graph, there would be considerable value in automating the search for  $\alpha$ . There exist newer flow algorithms [20] that can perform this search in time proportional to a single  $s$ - $t$  maximum flow calculation; however, there is no implementation known to exist at this time.

## Acknowledgements

We would like to thank John Joseph Carrasco, Frans Coetzee, Daniel Fain, Lee Giles, Eric Glover, Steve Lawrence, Kevin Lang, David Pennock, Satish Rao, and Bob Tarjan for many helpful discussions that occurred over the course of this research and in the preparation of this chapter.

## References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.
2. A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286, 1999.
3. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proc. 7th Int. World Wide Web Conf.*, 1998.
4. F. R. K. Chung. *Spectral Graph Theory*. Regional conference series in mathematics, no. 92. Amer. Math. Soc., Providence, RI, 1996.
5. F. R. K. Chung and L. Lu. The average distance in random graphs with given expected degrees. *Proceedings of National Academy of Science*, 99:15879–15882, 2002.
6. D. Cohn and T. Hofmann. The missing link - a probabilistic model of document content and hypertext connectivity. In *Neural Information Processing Systems 13*, 2001.
7. B. Everitt. *Cluster analysis*. Halsted Press, New York, 1980.
8. D. Fasulo. An analysis of recent work on clustering algorithms. Technical Report UW-CSE-01-03-02, Univ. of Washington, 1999.
9. G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *Proc. 6th Int. Conf. on Knowledge Discovery and Data Mining*, pages 150–160, 2000.
10. G. W. Flake, S. Lawrence, C. L. Giles, and F. Coetzee. Self-organization of the web and identification of communities. *IEEE Computer*, 35(3):66–71, 2002.
11. G. W. Flake, D. M. Pennock, and D. C. Fain. The self-organized web: The yin to the semantic web's yang. *IEEE Intelligent Systems*, 2003. To appear.
12. G. W. Flake, R. E. Tarjan, and K. Tsioutsoulis. Graph clustering techniques based on minimum cut trees. Technical Report 2002-006, NEC Research Institute, Princeton, NJ, 2002.
13. G.W. Flake, R.E. Tarjan, and K. Tsioutsoulis. Graph clustering and minimum cut trees. *Internet Mathematics*, 2003. To appear.
14. L. R. Ford Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian J. Math.*, 8:399–404, 1956.
15. E. Garfield. *Citation Indexing: Its Theory and Application in Science*. Wiley, New York, 1979.

16. D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *Proc. 9th ACM Conf. on Hypertext and Hypermedia*, 1998.
17. C. L. Giles, K. Bollacker, and S. Lawrence. CiteSeer: An automatic citation indexing system. In Ian Witten, Rob Akscyn, and Frank M. Shipman III, editors, *Digital Libraries 98 - The Third ACM Conference on Digital Libraries*, pages 89–98, Pittsburgh, PA, June 23–26 1998. ACM Press.
18. A. V. Goldberg, E. Tardos, and R. E. Tarjan. Network flow algorithms. In B. Korte, L. Lovasz, H. J. Promel, and A. Schrijver, editors, *Paths, flows, and VLSI-layout*, volume 9 of *Algorithms and Combinatorics*, pages 101–164. Springer, Berlin Heidelberg New York, 1990.
19. R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, December 1961.
20. J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut of a graph. In *Proc. 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 165–174, 1992.
21. T. Haveliwala. Topic-sensitive PageRank. In *Proceedings of the Eleventh International World Wide Web Conference*, 2002.
22. A. K. Jain and R. C. Dubes. *Algorithms and Clustering Data*. Prentice-Hall, New Jersey, 1998.
23. I.T. Jolliffe. *Principal Component Analysis*. Springer, Berlin Heidelberg New York, 1986.
24. D. R. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, 1996.
25. M. Kessler. Bibliographic coupling between scientific papers. *American Documentation*, 14:10–25, 1963.
26. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, 1998.
27. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
28. J. M. Kleinberg. An impossibility theorem for clustering. In *Advances of Neural Information Processing Systems 15*, Cambridge, MA, 2003. MIT Press.
29. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. In *Proc. 8th Int. World Wide Web Conf.*, 1999.
30. S. Lawrence and C. L. Giles. Accessibility of information on the web. *Nature*, 400:107–109, 1999.
31. R. Lempel and S. Moran. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):387–401, 2000.
32. D. M. Pennock, G. W. Flake, S. Lawrence, E. J. Glover, and C. L. Giles. Winners don't take all: Characterizing the competition for links on the web. *Proceedings of the National Academy of Sciences*, 99(8):5207–5211, 2002.
33. H. Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *J. Am. Soc. for Inf. Sci.*, 24(4):265–269, 1973.
34. G. Strang. *Linear algebra and its applications*. Harcourt Brace Jovanovich, San Diego, 1980.
35. K. Tsioutsoulouklis. *Maximum Flow Techniques for Network Clustering*. PhD thesis, Princeton University, Princeton, NJ, June 2002.
36. H. D. White and K. W. McCain. Bibliometrics. In *Ann. Rev. Info. Sci. and Technology*, pages 119–186. Elsevier, Amsterdam, 1989.



---

# Theory of Random Networks and Their Role in Communications Networks

José Fernando Mendes

Departamento de Física, Universidade de Aveiro,  
Campus Universitário de Santiago  
3810-193 Aveiro, Portugal  
jfmendes@fis.ua.pt

**Summary.** A short review of the recent developments in the statistical physics of random networks and their applications in many fields of science, namely in communications, is presented. The Internet and World Wide Web are perhaps two of the most impressive and intriguing examples of this type of networks. However, they are only particular examples of a more vast class of evolving networks. The present knowledge about their structure, topology, local properties and how they self-organize is very limited. The goal of this review is to show from a physicist's point of view the many problems addressed by such networks and the present understanding within the field. These networks present a rich set of scaling properties that can be characterized by the usual techniques of statistical physics. Most of the networks seen in nature as well as those generated by humans are of the scale-free type and present striking resilience properties against random attacks and damage, but they are vulnerable in the case of intentional attacks. Another important characteristic of such networks is the fact that the average distance between any two nodes is short even when its size is very large, which is known as the "small-world effect"

## 1 Introduction

The Internet and the World Wide Web (WWW) have experienced huge growth in recent years [31, 30, 6, 7, 43, 44, 38, 20, 60, 61, 64, 3, 77]. Their importance is related to the strong influence they have on our lives. Nevertheless, our knowledge about them is very limited. Given their youth, very little is known about their structure and hierarchical organization, their global topology, their local properties, and various other processes occurring within them. This knowledge is obviously necessary for effective functioning of the Internet and WWW. It can allow us, for example, to protect them against damage or attacks and to efficiently make use of all their capabilities.

Apart from the previous examples there are many other examples of evolving networks, e.g. collaboration networks [82, 69, 70], public relations networks [68], citations of scientific papers [41, 76], industrial networks [82], transportation networks [10], networks of relations between enterprises and agents in financial markets [65], biological networks [16, 9, 18, 79, 48] and so on. The fact that some of these networks

are finite in size introduces serious restrictions on extracting useful experimental data, namely because of strong size effects. The large size of the Internet and the WWW and their extensive and easily accessible documentation allows for reliable and informative experimental investigation of their structure and properties.

Only recently were features about the structural organization of such networks discovered [7, 43, 38, 20, 76, 48, 11, 12, 13, 29]. It has become clear that their complex structure is a natural consequence of the principles of their growth. Some models to describe such networks have been proposed [6].

The structure of random networks was initially studied by mathematicians; among them two names have special importance, the Hungarian mathematician Paul Erdős and his collaborator Alfred Rényi [36, 37]. However, they only studied equilibrium random graphs but not the most interesting type of networks, those that evolve into scale-free structures. Most of the results of graph theory [45] are related to the simplest random graphs with Poisson distributions of connections [36, 37] (classical random graphs).

Paul Baran [15] for the first time elaborated on the fundamental concepts of large communications networks. Many questions arise about them: What is the optimal topology of communications networks? What are the consequences of the topology on their stability and safety? These and many other vital problems were first studied by Baran in a practical context.

In the 1990s the Internet and the WWW achieved very large sizes, and continued to grow so rapidly that a number of search engines appeared with the aim to help us find documents in them. Each of these search engines only covers a small part of all the Web [60, 61, 19, 59, 21, 62, 58, 73, 57, 24, 2, 1]. More detailed knowledge of the structure of the WWW has become vitally important for its effective operation.

The first experimental data, mostly for the simplest structural characteristics of the communications networks, were obtained in 1997–1999 [7, 43, 44, 38, 57, 50]. Distributions of the number of connections in the networks and their surprisingly small average shortest path lengths were measured. An interesting property of these was observed, namely their long-tailed, power-law distributions. After these findings, physicists started intensive study of evolving networks in various areas, from communications to biology and public relations.

In this chapter some of the basic concepts and recent developments in the theory of random networks are presented. In Sect. 2 we present some simple characteristics of random networks. Section 3 is devoted to some examples of growing networks, like the WWW and Internet. In Sect. 4 some simple models are presented, namely the classical random graph of Erdős and Rényi, small-world networks and the Barabási–Albert model among others. In the last section we present some percolative properties of scale-free networks. A particular discussion is presented about the robustness and vulnerability of such networks.

## 2 Simple Characteristics of Networks

### 2.1 Adjacency Matrix

The networks that will be considered are graphs consisting of vertices (nodes) connected by edges (links). Edges may be directed or undirected. The structure of a network can be described by its adjacency matrix,  $\hat{A}$ , whose elements consist of zeros and ones. An element of the adjacency matrix of a network with undirected edges  $A_{ij}$  is 1 if vertices  $i$  and  $j$  are connected, and is 0 otherwise.

The adjacency matrix of a random network contains complete information about the structure of the net, and, in principle, one has to study just the adjacency matrix. Generally, this is not an easy task; however, only a very restricted set of structural characteristics are usually considered.

### 2.2 Degree

Degree is the simplest and the most studied characteristic. The degree  $k$  of a vertex is the total number of its connections. In physical literature, this quantity is often called “connectivity” that has a quite different meaning in graph theory. It can be obtained easily from the adjacency matrix. The degree of a node  $i$  is given by  $k_i = \sum_j A_{ij}$ .

### 2.3 Clustering Coefficient

For the description of connections in the environment closest to a vertex, the so-called *clustering coefficient* was introduced. For a network with undirected edges, the number of all possible connections of the nearest neighbours of a vertex  $i$  ( $z_i$  nearest neighbours) equals  $z_i(z_i - 1)/2$ ;  $y_i$  represents the number of them that are present. The clustering coefficient of this vertex,  $C_i \equiv y_i/[z_i(z_i - 1)/2]$ , is the ratio of existing connections between nearest neighbours of the vertex and the total number of possible connections. Averaging  $C_i$  over all vertices of a network yields the clustering coefficient of the network  $C$ . The clustering coefficient is the probability that two nearest neighbours of a vertex are nearest neighbours between themselves.

### 2.4 Shortest Path

It is possible to define the distance between two vertices,  $i$  and  $j$ , of a graph with unit length edges; it is the shortest path length  $\ell_{ij}$  from the vertex  $i$  to the vertex  $j$ . It is possible to introduce the distribution of the shortest path lengths between pairs of vertices of a network and the average shortest path length  $\bar{\ell}$  of a network (the average here is over all pairs of vertices between which a path exists and over all realizations of a network).

In a fully connected network  $\bar{\ell} = 1$ . One may roughly estimate  $\bar{\ell}$  of a network in which random vertices are connected. If the average number of nearest neighbours of a vertex is  $z_1$ , then about  $z_1^\ell$  vertices of the network are at a distance  $\ell$  from the vertex or closer. Hence,  $N \sim z_1^\ell$  and then  $\bar{\ell} \sim \ln N / \ln z_1$ , i.e. the average shortest path length value is small even for very large networks. This smallness is usually referred to as a *small-world effect* [82].

## 2.5 The Giant Component

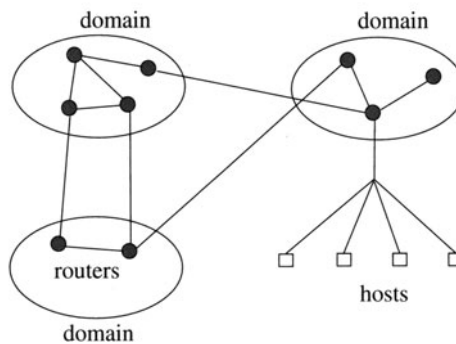
In general, a network consists of many disconnected parts. In networks with undirected edges, it is easy to introduce the notion of percolating clusters in the case of disordered lattices. If the relative size of the largest connected cluster of vertices of a network (the largest connected component) approaches a nonzero value when the network size tends to infinity, the system is above the percolating threshold, and this cluster is called the *giant connected component* of the network. In this case, the size of the next largest clusters, are small compared to the giant connected component for a large enough network. The size of this small components is of order  $\log N$ , where  $N$  is the network size.

One may generalize this notion for networks with directed edges. In this case, we have to consider a cluster of vertices from which one can approach any vertex of this cluster. Such a cluster may be called the strongly connected component. If the largest strongly connected component contains a finite fraction of all vertices in the large network limit, it is called the *giant strongly connected component*. Connected clusters obtained from a directed network by ignoring directions of its edges are called weakly connected components, and one can define the *giant weakly connected component* of a network.

## 3 Growing Networks in Nature

### 3.1 The WWW and the Internet

Roughly speaking, the Internet is a net of interconnected vertices: hosts (computers of users), servers (computers or programs providing a network service that also may be hosts), and routers that arrange traffic across the Internet (see Fig. 1). Connections are undirected, and traffic (including its direction) changes all the time. Routers are grouped in domains.



**Fig. 1.** Simple scheme of the structure of the Internet [38]

The World Wide Web is the array of its documents plus hyperlinks. Although hyperlinks are directed, pairs of counter links, in principle, may produce undirected connections.

### The Structure of the Internet

The Internet (the backbone support of the WWW) can be thought of as a set of vertices, linked by wires. The vertices of the Internet are hosts (computers of users), servers (computers or programs providing a network service that also may be hosts), and routers, which arrange traffic across the Internet. Connections are naturally undirected. (In an undirected network, the physical connection (wires) can transport information in both sides.) In January 2001, the Internet contained about 100 millions hosts. The structure of the Internet is not determined by its hosts, but rather by its routers and domains. In July 2000, there were about 150,000 routers in the Internet [42]. Soon, this number grew to 228,265 (data from [83]). Thus, one can consider the topology of the Internet on a router level or interdomain topology [38]. In the latter case (interdomain level), it is actually a small network, which does not allow us to perform a good analysis (see Table 1).

The last data of [38] are for December 1998. However, one may use more recent data on “autonomous systems”. Extensive data on connections of operating Autonomous Systems (AS) in the Internet are being collected by the National Laboratory for Applied Network Research (NLNR). For nearly each day, starting from November 1997, the NLNR has a map of connections of AS. These maps are closely related to the Internet graph on the interdomain level. Statistical analysis of these data was made in [75]. Table 1 summarises some of the known values for the Internet size over time. We see that the average degree of the Internet on the interdomain level (more rigorously speaking, on the AS level) is increasing.

date	$N_{\text{nodes}}$	$N_{\text{links}}$	$\bar{k}$
Nov 1997	3015	5156	3.42
Apr 1998	3530	6432	3.64
Dec 1998	4389	8256	3.76
Dec 1999	6374	13641	4.28
Sep 2001	11927	27492	4.61

**Table 1.** Known values of the Internet size (interdomain level) at different times

As was indicated above, most of connections in the Internet emerge between already existing sites. If the process of attachment of these edges is preferential, strongly connected sites usually have strongly connected nearest neighbours, unlike what was observed in [75]. One difficulty is that vertices in the Internet are of at least of two

distinct kinds. In [75], the difference between “stub” and “transit domains” of the Internet is noticed. Stub domains have no connections between them and connect to transit domains, which are, in contrast, well interconnected. Therefore, new connections or rewiring are not possible between all vertices. This may be the reason for the observed correlations. A different classification of the Internet sites was used in [22]. The vertices of the Internet were separated into two groups, namely “users” and “providers”. Interaction between these two kinds of sites leads to the self-organization of the growing network into a scale-free structure.

The process of the attachment of new edges in these maps of Internet was empirically studied in [47]. It was found that the probability that a new edge is attached to a vertex is a linear function of the vertex degree.

A very important feature of the Internet, both on the AS (or the interdomain) level and on the level of routers, is that its vertices are physically attached to specific places in the world and have fixed geographic coordinates. The geographic locations of vertices and the distribution of Euclidean distances are essential for the resulting structure of the Internet. This factor was studied and modelled in a recent paper [83]. It was observed that routers and AS correlate with the population density. All three sets—population, router, and AS space densities—form fractal structures in space. The fractal dimension of these fractals was found to be approximately 1.5 (from data for North America). Maps of AS and the map of 228, 265 routers were analysed. In particular, the average shortest distance between two routers was found to be approximately 9 [83].

### The Structure of the WWW

The WWW consists of a set of documents (pages) plus hyperlinks between them. The WWW in contrast to the Internet, is a directed network [56]. Although hyperlinks are directed, pairs of counter-links, in principle, may produce undirected connections. Links inside pages (self-references) are usually not considered as edges of the WWW, so this network does not contain “tadpoles” (closed one-edge loops). Figure 2 shows in schematic form the structure of a directed graph and all its components (for a detailed description, see [32]).

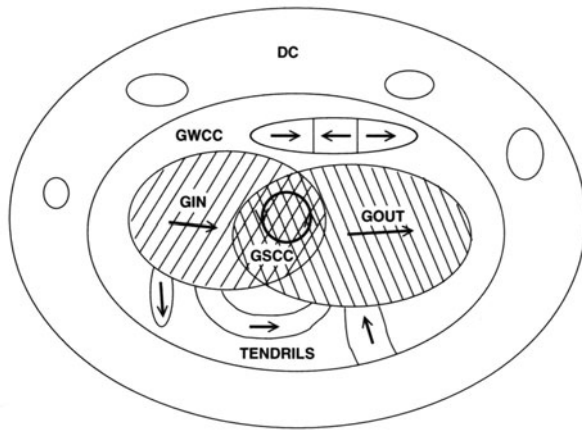
Date	$N_{\text{nodes}}$	$N_{\text{links}}$	$\bar{k}$
May 1999	$203 \times 10^6$	$1466 \times 10^6$	7.22
Oct 1999	$271 \times 10^6$	$2130 \times 10^6$	7.85

**Table 2.** Known values of the size of the WWW at different times

According to [20], in May 1999, and using data from Altavista, the WWW consisted of  $203 \times 10^6$  vertices (URLs, i.e. pages) and  $1466 \times 10^6$  hyperlinks. So, the average in- and out- degree were  $\bar{k}_{\text{in}} = \bar{k}_{\text{out}} = 7.22$ . It is possible to find values for

the WWW size for other times. The average in- and out-degrees are equal to each other, since all the connections are inside the WWW. As we can see from the Table 2, the average degree of the WWW is increasing.

If we discard the network part consisting of the disconnected clusters, that is, Disconnected Components (DC), we get the GWCC. The GWCC can be divided into four components: the GSCC—from each vertex of the GSCC, there exists a directed path to any other its vertex; the *giant out-component* (GOUT)—the vertices that are reachable from the GSCC by a directed path, so that GOUT includes GSCC; the *giant in-component* (GIN)—the vertices from which one can reach the GSCC by a directed path so that GIN includes GSCC; and the *tendrils* (TE)—the rest of the GWCC. This part consists of the vertices that have no access to the GSCC and are not reachable from it.



**Fig. 2.** Structure of a directed graph when the giant strongly connected component is present [25] (see text). If one ignores the directedness of edges, the network consists of the *giant weakly connected component* (GWCC) – actually, the usual percolating cluster – and disconnected components (DC). Accounting for the directedness of edges, the GWCC contains the following components: (a) the *giant strongly connected component* (GSCC), that is, the set of vertices reachable from its every vertex by a directed path; (b) the *giant out-component* (GOUT), the set of vertices approachable from the GSCC by a directed path (includes the GSCC); (c) the *giant in-component* (GIN), contains all vertices from which the GSCC is approachable (includes the GSCC); (d) the *tendrils*, the rest of the GSCC, i.e. the vertices which have no access to the GSCC and are not reachable from it. In particular, this part includes something like “tendrils” [20] but there are also “tubes” and numerous clusters that are only “weakly” connected

Using the previous definitions, one can say that:

Network = GWCC + DC and GWCC = GIN + GOUT – GSCC + TE.

According to [20], in May 1999, the entire Web contained  $203 \times 10^6$  pages, divided in the following way: the GWCC,  $186 \times 10^6$  pages (91% of the total number of pages); and the DC,  $17 \times 10^6$  pages. In turn, the GWCC included the GSCC,

$56 \times 10^6$  pages; the GIN,  $99 \times 10^6$  pages; the GOUT,  $99 \times 10^6$  pages; and the TE,  $44 \times 10^6$  pages. Both distributions of the sizes of strongly connected components and of the sizes of weakly connected ones were fitted by power-law dependencies with exponents of approximately 2.5.

The probability that a directed path is present between two random vertices was estimated as 24%. For pairs of pages of the WWW between which directed paths exist, the *average shortest directed path length* equals 16. For pairs between which at least one undirected path exists, the *average shortest undirected path length* equals 7.

The value of the average shortest directed path length estimated from data extracted from the nd.edu domain of the WWW was 19 [7]. Its size dependence was estimated as  $\bar{\ell}(N) \approx 0.35 + 2.06 \log N$ . The value of  $\bar{\ell}(N)$  was extrapolated to  $N = 800,000,000$ , that is, the estimation of the size of the WWW in 1999. The result, i.e.  $\bar{\ell}(800,000,000) \approx 19$ , is very close to the above-cited value  $\bar{\ell}(200,000,000) = 16$  of [20] if one accounts for the difference in sizes.

The maximum shortest path between nodes belonging to the GSCC equals 28. The maximum shortest directed path for nodes of the WWW between which a directed path exists is greater than 500 (some estimates indicate that it may be even 1000). Although the GSCC of the WWW is rather small, most pages of the WWW belong to the GWCC. Furthermore, even if all links to pages with in-degree larger than 2 are removed, the GWCC does not disappear. This is clearly demonstrated by the data of [20].

The size of the GWCC (from Altavista, May 1999) is  $186 \times 10^6$  pages. If all in-links to pages with  $k_i \geq k_i^{(\max)} = 1000, 100, 10, 5, 4$ , and 3 are removed, the size of the retaining GWCC is  $177 \times 10^6, 167 \times 10^6, 105 \times 10^6, 59 \times 10^6, 41 \times 10^6$ , and  $15 \times 10^6$  pages, respectively. So, for  $k_{in}^{(\max)} = 100$ , the corresponding size is  $\text{GWCC} = 167 \times 10^6$  pages; for  $k_{in}^{(\max)} = 10$ ,  $\text{GWCC} = 105 \times 10^6$  pages; for  $k_{in}^{(\max)} = 5$ ,  $\text{GWCC} = 59 \times 10^6$  pages; for  $k_{in}^{(\max)} = 4$ ,  $\text{GWCC} = 41 \times 10^6$  pages; for  $k_{in}^{(\max)} = 3$ ,  $\text{GWCC} = 15 \times 10^6$  pages.

## 4 The Models

### 4.1 Classical Random Graphs, the Erdős–Rényi model

The simplest and most studied network with undirected edges was introduced by Erdős and Rényi (ER model) [36, 37]. In this network the total number of vertices  $N$  is fixed, and the probability that two arbitrary vertices are connected is  $p$ .

In this case, the average number of edges is given by  $pN(N-1)/2$  and the degree distribution obeys a binomial distribution,

$$P(k) = \binom{N-1}{k} p^k (1-p)^{N-1-k}. \quad (1)$$



The average degree can be calculated easily and is given by  $\bar{k} \approx pN$ . For large  $N$ , the distribution in Eq. (1) is Poisson-like:

$$P(k) = e^{-\bar{k}} \bar{k}^k / k!. \quad (2)$$

Therefore, the distribution rapidly decreases at large degrees. Such distributions are characteristic for classical random networks. Moreover, in the mathematical literature, the term “random graph” usually means a network with a Poisson degree distribution and statistically uncorrelated vertices. Here, we prefer to call it “classical random graph”.

We have already presented the estimate for an average shortest path length of this network,  $\bar{\ell} \sim \ln N / \ln[pN]$ . At small values of  $p$ , the system consists of small clusters. At large  $N$  and large enough  $p$ , the giant connected component appears in the network. The percolation threshold is  $p_c \cong 1/N$ , that is,  $\bar{k}_c = 1$ . In fact, the ER model describes percolation on a lattice of infinite dimension, and the adequate mean-field description is possible.

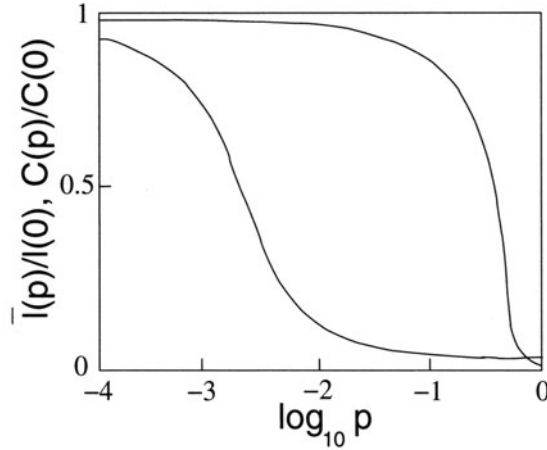
## 4.2 Small-World Networks (WS)

Most random networks show the so-called small-world effect, i.e. their average shortest path length is small. Watts and Strogatz [82] noticed a key feature in these networks: they found that the small shortest path is due to the long-range connections, while the short range links are responsible for a high clustering. Although the average shortest path length between their vertices is really small and is of the order of the logarithm of their size, the clustering coefficient is much greater than it should be for classical random graphs [27, 55]. They proposed a model (the WS model) that demonstrates such a possibility and also called it the small-world network. The model belongs to the class of networks displaying a crossover from ordered to random structures and may be treated analytically. By definition of Watts and Strogatz, the small-world networks are those with “small” average shortest path lengths and “large” clustering coefficients. In fact, the networks introduced by Watts and Strogatz have an important generic feature: they are constructed from ordered lattices by random rewiring of edges or by addition of connections between random vertices.

### The Watts–Strogatz Model

The original network of Watts and Strogatz is constructed in the following way. Initially, a regular one-dimensional lattice with periodical boundary conditions is present. Each of  $L$  vertices has  $z \geq 4$  nearest neighbours ( $z = 2$  was not appropriate for Watts and Strogatz since, in this case, the clustering coefficient of the original regular lattice is zero). Then one takes all the edges of the lattice in turn and with probability  $p$  rewires to randomly chosen vertices. In such a way, a number of far connections appears. Obviously, when  $p$  is small, the situation has to be close to the original regular lattice. For large enough  $p$ , the network is similar to the classical random graph.

Watts and Strogatz studied the crossover between these two limits. The main interest was in the average shortest path  $\bar{\ell}$  and the clustering coefficient (recall that each edge has unit length). The simple but exciting result was that even for a very small probability of rewiring, when the local properties of the network are still nearly the same as those of the original regular lattice and the clustering coefficient does not differ essentially from its initial value, the average shortest path length is of the order of the one observed for classical random graphs.



**Fig. 3.** Average shortest path length  $\bar{\ell}$  and clustering coefficient  $C$  of the Watts–Strogatz model vs. fraction of the rewired links  $p$  [82]. Both are normalized to their values for the original regular lattice ( $p = 0$ ). The network has 1000 nodes. The average number of the nearest neighbors equals 10.  $C$  is practically constant in the range where  $\bar{\ell}$  sharply diminishes

This result seems quite natural. Indeed, the average shortest path length is very sensitive to shortcuts. One can see that it is enough to make a few random rewirings to decrease  $\bar{\ell}$  by several times. On the other hand, several rewired edges cannot crucially change the local properties of the entire network. This means that the global properties of the network change strongly even at  $pzL \sim 1$ , when there is one shortcut in the network, i.e. at  $p \sim 1/(Lz)$ , when the local characteristics are still close to the regular lattice.

Recall that the simplest local characteristic of networks is their degree. Hence, it would be natural to compare, at first, the behavior of  $\bar{\ell}$  and  $\bar{k}$ . However, in the originally formulated WS model,  $\bar{k}$  is independent of  $p$  since the total number of edges is conserved during the rewiring. Watts and Strogatz took another characteristic for comparison, the characteristic of the closest environment of a vertex, i.e., the clustering coefficient  $C$ .

Using the rewiring procedure, a network with a small average shortest path length and a large clustering coefficient was constructed. Instead of the rewiring of edges, one can add shortcuts to a regular lattice [72]. The main features of the model do not

change. One can also start with a regular lattice of an arbitrary dimension  $d$  where the number of vertices  $N = L^d$  [49, 51]. In this case, the number of edges in the regular lattice is  $zL^d/2$ . To keep the correspondence to the WS model, let us define  $p$  in such a way that for  $p = 1$ ,  $zL^d/2$  random shortcuts are added. Then, the average number of shortcuts in the network is  $N_s = pzL^d/2$ . At small  $N_s$ , there are only two natural lengths in the system,  $\bar{\ell}$  and  $L$ , since the lattice spacing is not important in this regime. Thus, their dimensionless ratio can be only a function of  $N_s$ ,

$$\frac{\bar{\ell}}{L} = f(2N_s) = f(pzL^d), \quad (3)$$

where  $f(0) \sim 1$  for the original regular lattice and  $f(x \gg 1) \sim \ln x/x^{1/d}$ . From Eq. (3), one can immediately obtain the following relation,  $\bar{\ell}(pz)^{1/d} = \Phi(L(pz)^{1/d})$ . Here,  $\xi = (pz)^{-1/d}$  has the meaning of length:  $N_s \xi^d \sim L^d$ , which is the average distance between the closest end points of shortcuts measured on the regular lattice. In fact, one must study the limit  $L \rightarrow \infty$ ,  $p \rightarrow 0$ , as the number of shortcuts  $N_s = pzL^d/2$  is fixed. The last relation for  $\bar{\ell}$ , in the case where  $d = 1$ , was proposed and studied by simulations and afterwards analytically [17].

### 4.3 Growing Exponential Networks

The case considered before had a fixed number of vertices. Let us discuss the simplest random network in which the number of vertices increases [11]. At each increment of time let a new vertex be added to the network. It connects to a randomly chosen (i.e. *without any preference*) old vertex. We will consider that connections are undirected, however, without loss of generality. The growth begins from the configuration consisting of two connected vertices at time  $t = 1$ , so at time  $t$  the network consists of  $t + 1$  vertices and  $t$  edges. The total degree at time  $t$  equals  $2t$ . One can check that the average shortest path length in this network is  $\bar{\ell} \sim \ln t$ , like in classical random graphs.

It is easy to obtain the degree distribution for such a net. We may label vertices by their birth times,  $s = 0, 1, 2, \dots, t$ . Let  $p(k, s, t)$  be the probability that a vertex  $s$  has degree  $k$  at time  $t$ . The master equation describing the evolution of the degree distribution of individual vertices is

$$p(k, s, t + 1) = \frac{1}{t + 1} p(k - 1, s, t) + \left(1 - \frac{1}{t + 1}\right) p(k, s, t), \quad (4)$$

$p(k, s = 0, 1, t = 1) = \delta_{k,1}$ ,  $\delta(k, s = t, t \geq 1) = \delta_{k,1}$ . The two terms correspond to the following possibilities for a vertex  $s$ : (i) with probability  $1/(t + 1)$ , it may get an extra edge from the new vertex and increase its own degree by 1; (ii) with the complimentary probability  $1 - 1/(t + 1)$  the vertex  $s$  may remain in the former state with the former degree.

Let us define the total degree distribution of the entire network by:

$$P(k, t) = \frac{1}{t+1} \sum_{s=0}^t p(k, s, t). \quad (5)$$

The stationary solution, i.e. at  $t \rightarrow \infty$ ,  $P(k) \equiv P(k, t \rightarrow \infty)$  can be obtained after some straightforward analysis, resulting in an exponential form:

$$P(k) = 2^{-k}. \quad (6)$$

These networks are usually called “exponential”. This form differs from the Poisson degree distribution of classical random graphs. Nevertheless, both distributions are rapidly decreasing functions, unlike degree distributions of numerous large networks in nature.

The average degree of vertex  $s$  at time  $t$  is

$$\bar{k}(s, t) = \sum_{k=1}^{\infty} k p(k, s, t). \quad (7)$$

Applying  $\sum_{k=1}^{\infty} k$  to both sides of Eq. (4), we get the equation for this quantity,

$$\bar{k}(s, t+1) = \bar{k}(s, t) + \frac{1}{t+1}. \quad (8)$$

For  $s, t \gg 1$ , the following asymptotic form appears,  $\bar{k}(s, t) = 1 - \ln(s/t)$ , i.e. the average degree of individual vertices of this network *weakly* diverges in the region of the oldest vertex. Hence, the oldest vertex is the “most” connected.

From Eq. (4), one can also find the degree distribution of individual vertices,  $p(k, s, t)$ , for large  $s$  and  $t$  and fixed  $s/t$ :

$$p(k, s, t) = \frac{s}{t} \frac{1}{(k+1)!} \ln^{k+1} \left( \frac{t}{s} \right). \quad (9)$$

One sees that this function decreases rapidly at large values of degree  $k$ . Similar results may be easily obtained for a network in which each new vertex has not one, as previously, but any fixed number of connections with randomly chosen old vertices.

#### 4.4 Scale-Free networks

Most of the large growing networks in Nature are scale-free, i.e., their degree distributions are of a power-law form. The natural question is how they self-organize into scale-free structures while growing. What is the mechanism responsible for such self-organization? To explain these phenomena, the idea of *preferential linking* (preferential attachment of edges to vertices) was proposed [11].

## Barabási-Albert Model: The Preferential Linking Concept

We have demonstrated that if new connections in a growing network appear between vertices chosen without any preference, e.g. between new vertices and randomly chosen old ones, the degree distribution is exponential. Nevertheless, in real networks, linking is very often *preferential*.

For example, when you make a new reference in your own page, the probability that you refer to a popular Web document is certainly higher than the probability that this reference is to some poorly known document to which no one has referred to before you. Therefore, popular vertices with high numbers of links are more attractive for new connections than vertices with few links, that is, *popularity is attractive*.

Let us demonstrate the growth of a network with preferential linking using, as the simplest example, the Barabási–Albert model (the BA model) [11]. Now a new vertex connects not to a randomly chosen old vertex but to a vertex chosen *preferentially* [28, 33, 54, 34].

We describe here the simplest situation: The probability that the edge is attached to an old vertex is proportional to the degree of this old vertex, i.e. to the total number of its connections. At time  $t$ , the total number of edges is  $t$ , and the total degree equals  $2t$ . Hence, this probability equals  $k/(2t)$ . One should emphasize that this is only a particular form of a *preference function*. However, just the linear type of the preference was indicated in several real networks. To account for preferential linking, we must make obvious modifications to the master equation, Eq. (4). For the BA model, the master equation takes the following form:

$$p(k, s, t + 1) = \frac{k - 1}{2t} p(k - 1, s, t) + \left(1 - \frac{k}{2t}\right) p(k, s, t), \quad (10)$$

with the initial condition  $p(k, s = 0, 1, t = 1) = \delta_{k,1}$  and the boundary condition  $p(k, t, t) = \delta_{k,1}$ . From Eqs. (5) and (10), we get the master equation for the total degree distribution,

$$(t + 1)P(k, t + 1) - tP(k, t) = \frac{1}{2}[(k - 1)P(k - 1, t) - kP(k, t)] + \delta_{k,1} \quad (11)$$

and, in the limit  $t \rightarrow \infty$ , the equation for the stationary distribution,

$$P(k) + \frac{1}{2}[kP(k) - (k - 1)P(k - 1)] = \delta_{k,1}. \quad (12)$$

In the continuum  $k$  limit, this equation is of the form  $P(k) + (1/2)d[kP(k)]/dk = 0$ . The solution of the last equation is  $P(k) \propto k^{-3}$ . Thus, the preferential linking of the form that we consider provides a scale-free network, and the  $\gamma$  exponent of its distribution equals 3 [11, 12]. This value is exact, see [34] and the discussion below.

We emphasize that the preferential linking mechanism [11] is the basic idea of the modern theory of evolving networks. The recent empirical data [47] on the dynamics of the attachment of new edges in various growing networks provides support for this mechanism.

### Scaling Relations and Cutoff

It can be shown that a certain number of quantities of particular scale-free networks may be written in a scaling form, and the scaling exponents involved are connected by a simple relation. To get the scaling properties, a continuum treatment is sufficient [35, 26], so that we can use the following expressions:

$$P(k, t) = \frac{1}{t} \int_{t_0}^t ds p(k, s, t), \quad (13)$$

$$\bar{k}(s, t) = \int_0^\infty dk k p(k, s, t). \quad (14)$$

In addition, we will need the normalization condition for  $p(k, s, t)$ ,

$$\int_0^\infty dk p(k, s, t) = 1. \quad (15)$$

If the stationary distribution exists, then from Eq. (13), it follows that  $p(k, s, t)$  has to be of the form  $p(k, s, t) = \rho(k, s/t)$ . From the normalization condition, Eq. (15), we get  $\int_0^\infty dk \rho(k, x) = 1$ , so  $\rho(k, x) = g(x)f(kg(x))$ , where  $g(x)$  and  $f(x)$  are arbitrary functions.

Let us assume that the stationary distribution  $P(k)$  and the average degree  $\bar{k}(s, t)$  exhibit scaling behavior, that is,  $P(k) \propto k^{-\gamma}$  for large  $k$  and  $\bar{k}(s, t) \propto s^{-\beta}$  for  $1 \ll s \ll t$ . Then, from Eq. (14), one sees that  $\int_0^\infty dk k \rho(k, x) \propto x^{-\beta}$ . Substituting  $\rho(k, x)$  into this relation, one obtains  $g(x) \propto x^\beta$ . Of course, without loss of generality, one may set  $g(x) = x^\beta$ , so that we obtain the following scaling form of the degree distribution of individual vertices,

$$p(k, s, t) = (s/t)^\beta f(k(s/t)^\beta). \quad (16)$$

Finally, assuming the scaling behavior of  $P(k)$ , i.e.  $\int_0^\infty dx \rho(k, x) \propto k^{-\gamma}$ , and using Eq. (16), we obtain  $\gamma = 1 + 1/\beta$ , i.e. this relation between the exponents is universal for scale-free networks. Here we used the rapid convergence of  $\rho(k, x)$  at large  $x$ .

Now we can discuss the size effects in growing scale-free networks. Accounting for the rapid decrease of the function  $f(z)$  in Eq. (16), one sees that the power-law dependence of the total degree distribution has a cutoff at the characteristic value,

$$k_{\text{cut}} \sim t^\beta = t^{1/(\gamma-1)}. \quad (17)$$

In fact,  $k_{\text{cut}}$  is the generic scale of all “scale-free” networks. It also follows from the condition  $t \int_{k_{\text{cut}}}^\infty dk P(k) \sim 1$ , i.e.  $t \int_{k_{\text{cut}}}^\infty dk k^{-\gamma} \sim 1$ . This means that only one vertex in a network has degree above the cutoff. A more precise estimate is  $k/k_0 \sim t^{1/(\gamma-1)}$ , where  $k_0$  is the lower boundary of the power-law region of the degree distribution. The cutoff sets strong restrictions for observations of power-law distributions since there are few really large networks in nature.

No scale-free networks with large values of  $\gamma$  were observed. The reason for this is clear. Indeed, the power-law dependence of the degree distribution can be observed

only if it exists for at least 2 or 3 decades of degree. For this, the networks have to be large: their size should be at least  $t > 10^{2.5(\gamma-1)}$ . Then, if  $\gamma$  is large, one practically has no chance to find scale-free behavior.

In Fig. 4, in the log-linear scale, we present the values of the  $\gamma$  exponents of all the networks reported as having power-law degree distributions versus their sizes. One sees that almost all the plotted points are inside the region restricted by the lines:  $\gamma = 2$ ,  $\log_{10} t \sim 2.5(\gamma - 1)$ , and by the logarithm of the size of the largest scale-free network, the World-Wide Web,  $\log_{10} t \sim 9$ .

## 5 Percolation on Networks

Rigorously speaking, percolation is a phenomenon determined for structures with well-defined metric structure, e.g. regular lattices. In the case of networks, where it is hard to introduce metric coordinates, one can speak about the emergence of a giant component. In physical literature, a phenomenon related to the emergence of a giant component in networks is usually called a *percolating cluster*, and the phase transition associated with the emergence of the giant component is called the *percolation threshold* [71, 66, 67].

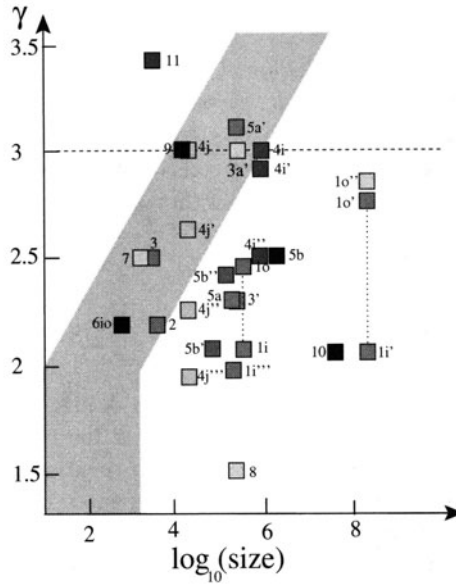
If the giant component is absent, the network is only a set of small clusters, so that the study of this characteristic is of primary importance. For regular lattices, to observe the percolation phenomenon one must remove a fraction of sites or bonds. In the case of networks it is not necessary to delete vertices or edges to eliminate their giant components. For instance, one can approach the percolation threshold changing the degree distribution of a network.

One should note that percolation phenomena in equilibrium and evolving (growing) networks are of different natures, so hereafter we consider them separately. Furthermore, the existing percolation theory for equilibrium networks [71] and its generalizations are valid only for specific graphs constructed according the Molloy-Reed criterion [66, 67].

### 5.1 Failures and Attacks

The effect of random damage and attack on communications networks (the WWW and Internet) was simulated by Albert et al. [8]. Failures (random damage) were modelled by the instant removal of a fraction of randomly chosen vertices. The intentional damage (attack) was described by the instant deletion of a fraction of vertices with the highest numbers of connections (degree). The networks were grown and then were instantly damaged. In these simulations, the networks were treated as undirected. The following quantities were measured as functions of the fraction  $f$  of deleted vertices:

- the average shortest path  $\bar{\ell}$  between randomly chosen vertices of the network,
- the relative size  $S$  of the largest connected component (corresponds to the giant connected component if it exists), and
- the average size  $\bar{s}$  of connected components (excluding the giant connected one).



**Fig. 4.** Log-linear plot of the  $\gamma$  exponents of all networks reported as presenting power-law (in-, out-) degree distributions (i.e. scale-free networks) vs. their sizes. The line  $\gamma \sim 1 + \log_{10} t/2.5$  is the estimate of the finite size boundary for the observation of the power-law degree distributions for  $\gamma > 2$ . Here 2.5 is the range of degrees (orders) that we believe is necessary to observe a power law. The *dashed line*,  $\gamma = 3$ , is the resilience boundary. This boundary is important for networks which must be stable to random breakdowns. There exists a chance that some of these networks are actually not in the class of scale-free networks.

The points  $1i$  and  $1o$  are obtained from in- and out-degree distributions of the complete map of the nd.edu domain of the WWW [7];  $1i'$  and  $1o'$  are from in- and out-degree distributions of the pages of the WWW scanned by Altavista in October 1999 [20, 57];  $1o''$  is the  $\gamma_o$  value from another fitting of the same data [71];  $1i'''$  is  $\gamma_i$  for domain level of the WWW in the spring of 1997 [2];  $2$  is  $\gamma$  for the interdomain level of the Internet in December 1998 [38];  $2'$  is  $\gamma$  for the network of operating AS for one day in December 1999 [75];  $3$  is  $\gamma$  for the router level of the Internet in 1995 [38];  $3'$  is  $\gamma$  for the router level of the Internet in 2000 [42];  $4i$  is  $\gamma_i$  for citations of the ISI database from 1981 to June 1997 [76];  $4i'$  is the result of the different fitting of the same data [78];  $4i''$  is another estimate obtained from the same data [53, 52];  $4j$  is  $\gamma_i$  for citations of Phys. Rev. D **11-50** (1975–1994) [76];  $4j'$  is a different fitting of the same data [78];  $4j''$  is another estimate from the same data [53, 52];  $4j'''$  is  $\gamma_i$  for citations of Phys. Rev. D from 1982 to June 1997 [80];  $5a$  is the  $\gamma$  exponent for the collaboration network of movie actors [11];  $5a'$  is the result of another fitting for the same data [5];  $5b$  is  $\gamma$  for the collaboration network of MEDLINE [69];  $5b'$  is  $\gamma$  for the collaboration net collected from mathematical journals [14];  $5b''$  is  $\gamma$  for the collaboration net collected from neuroscience journals [14];  $6io$  is  $\gamma_i = \gamma_o$  for networks of metabolic reactions [48];  $7$  is  $\gamma$  of the network of protein-protein interactions (yeast proteome) if it is treated as undirected [46, 81];  $8$  is  $\gamma$  of the degree distribution of the word Web in the range below the crossover point [40];  $9$  is  $\gamma$  of large digital electronic circuits [39];  $10$  is  $\gamma_i$  of the telephone call graph [4] (the out-degree distribution of this graph cannot be fitted by a power-law dependence);  $11$  is  $\gamma$  of vertices in the Web of human sexual contacts [63]



A striking difference between scale-free networks and exponential ones was observed. Whereas the exponential network produces the same dependencies  $\bar{\ell}(f)$ ,  $S(f)$ , and  $\bar{s}(f)$  for both kinds of damage, for all scale-free networks that are discussed here, these curves are distinct for different types of damage. The qualitative effect of the intentional damage was more or less the same for all four networks (Fig. 5). The average shortest path rapidly grows with growing  $f$ , and the size of the giant connected component becomes zero at some point  $f_c$ , indicating the percolation threshold,  $S(f_c) = 0$ . Near this point,  $S(f) \propto (f_c - f)$  as in the mean-field theory of percolation. At  $f_c$ ,  $\bar{s}$  diverges. Hence, the behavior is usual for the mean-field (or infinitely dimensional) percolation and for the percolation in the classical random graphs. Nevertheless, one general distinct feature of these scale-free networks should be emphasized. The value  $f_c$  in them is anomalously low, that is, several percent, unlike the percolation threshold of the exponential networks, so that such networks are very sensitive to intentional damage [74, 64].

The main observation of [8] is that random damage has a far less pronounced effect on scale-free networks than intentional damage. The variations of the average shortest distance with  $f$  are hardly visible. The size of the giant strongly connected component decreases slowly until it disappears in the vicinity of  $f = 1$ , while  $\bar{s}(f)$  grows smoothly with growing  $f$  without visible signs of singularity. This means that these scale-free networks are extremely resilient to random damage. To destroy them acting in such away, that is, to eliminate their giant connected component and to disintegrate them to a set of uncoupled clusters, it is necessary to delete practically all their vertices!

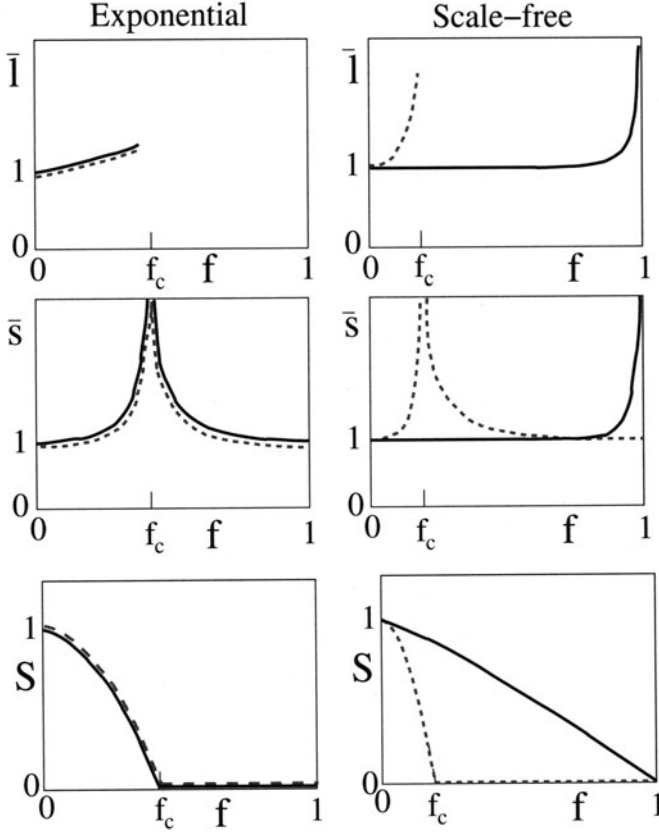
Similar observations were made for scale-free networks of metabolic reactions, protein networks, and food Webs.

The effect of attacks on scale-free networks seems rather natural since vertices of the highest degree determine the structure of these networks, but the vitally important resilience against failures needs detailed explanation. Several recent papers have been devoted to the study of this intriguing problem.

## 5.2 Resilience Against Random Breakdowns

As we saw in Sect. 5.1, the random breakdowns (failures) of networks more or less correspond to the classical site percolation problem, i.e. a vertex of the network is present with probability  $p = 1 - f$ , and one has to study how properties of the network vary with changing  $p$ . Now we have at hand the controlling parameter  $p$  to approach the percolation threshold. The first calculations of the threshold for failures in scale-free networks have been done by Cohen et al. [23].

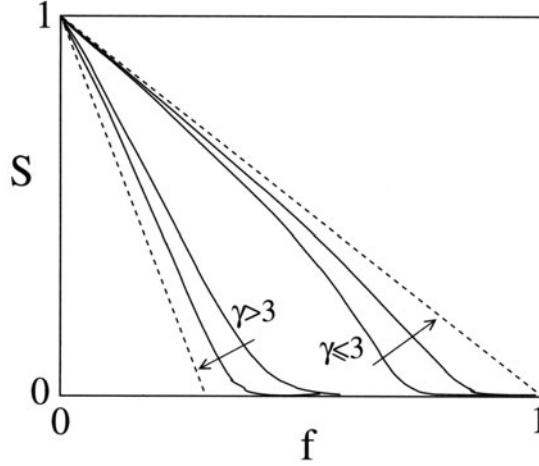
For networks with power-law distributions, the variation of the size of the giant connected component with  $p$  was studied through simulation (Fig. 6). Strong size effects was observed. For  $\gamma > 3$ , the percolation threshold is visible at the point  $p_c = 1 - f_c > 0$ . For  $\gamma \leq 3$ , if a network is infinite, one sees that  $p_c$  approaches zero, so that in this situation, one has to remove (at random) practically all vertices of the network to eliminate the giant connected component!



**Fig. 5.** Schematic plots of the effect of intentional and random damage (attack and failures) on the characteristics of exponential undirected networks and scale-free undirected ones with exponent  $\gamma \leq 3$  [8]. The average shortest path between vertices  $\bar{l}$ , the size of the largest connected component  $S$ , and the average size of isolated clusters,  $\bar{S}$  are plotted vs. the fraction of removed vertices  $f \equiv 1 - p$ . The networks are large. The *solid lines* show the effect of random damage; the effect of the intentional damage is shown by the *dashed lines*. For exponential networks, both kinds of damage produce the same dependencies. For scale-free networks with  $\gamma \leq 3$ , in the event of the random damage, the percolation threshold is at the point  $f \rightarrow 1$ .

The condition  $\gamma \leq 3$  for the resilience of scale-free networks to random damage and failures makes the values of the exponents of the degree distributions of communications networks quite natural. All of them are less than 3 (Sect. 3.1). Note that many other networks, e.g. biological ones, must necessarily be resilient to failures. Therefore, this condition is of great importance.

One should note that this result is valid for infinite networks. As one can see from Fig. 6, for  $\gamma < 3$  the size effects are very strong, and the curves slowly approach the infinite network limit where  $p_c(N \rightarrow \infty) = 0$ . Here,  $N$  is the size of the network. Let



**Fig. 6.** Schematic plot of dependences of the relative size of the largest connected component in the randomly damaged finite size net vs. the fraction of removed vertices  $f = 1 - p$  [23]. Distinct curves correspond to different network sizes  $N$  and two values of the  $\gamma$  exponent,  $2 < \gamma \leq 3$  and  $\gamma > 3$ . Arrows show displacement of the curves with increasing  $N$ . Dashed lines depict the limits  $N \rightarrow \infty$ . Notice the strong size effects

us estimate this size effect for  $2 < \gamma \leq 3$  by introducing the size-dependent threshold  $p_c(N)$ , whose meaning is clear from Fig. 6. When  $N \gg 1$ , one obtains

$$p_c(N) = \frac{z_1}{z_2} \cong \frac{\bar{k}}{k^2} \approx \frac{\int_{k_0}^{N^{1/(\gamma-1)}} dk k k^{-\gamma}}{\int_{k_0}^{N^{1/(\gamma-1)}} dk k^2 k^{-\gamma}}. \quad (18)$$

Notice that, for  $2 < \gamma \leq 3$ , the average number of second-nearest neighbors is  $z_2 \cong \bar{k}^2$ , since the second moment diverges as  $N \rightarrow \infty$ . The nature of the upper cutoff of the power-law degree distribution,  $k_{\text{cut}}/k_0 \sim N^{1/(\gamma-1)}$ , was explained in Sect. 4.4, and  $k_0 > 0$  can be estimated as the minimal value of the degree in the network. One may expect that  $k_0 \sim 1$ .

If  $2 < \gamma < 3$ , from Eq. (18) it readily follows that

$$p_c(N) = C(k_0, \gamma) N^{-(3-\gamma)/(\gamma-1)}. \quad (19)$$

Here,  $C(k_0, \gamma)$  does not depend on  $N$  and is of the order of 1. The value of  $C(k_0, \gamma)$  actually depends on the particular form of the degree distribution for small values of degree and is not of great interest here. When  $\gamma$  is close to 3,  $p_c = 0$  can be approached only for a huge network. Even if  $\gamma = 2.5$  and a net is very large, we get noticeable values of the threshold  $p_c$ , e.g.,  $p_c(N = 10^6) \sim 10^{-2}$  and  $p_c(N = 10^9) \sim 10^{-3}$ .

This finite size effect is most pronounced when  $\gamma = 3$ . In this case, Eq. (18) gives

$$p_c(N) \approx \frac{2}{k_0 \ln N}. \quad (20)$$

For example, if  $k_0 = 3$ ,  $p_c(N = 10^4) \approx 0.07$ ,  $p_c(N = 10^6) \approx 0.05$ , and  $p_c(N = 10^9) \approx 0.03$ .

These estimates demonstrate that in reality, that is, for *finite* scale-free networks, the percolation threshold is actually present even if  $2 < \gamma \leq 3$  (Fig. 6). Only if  $\gamma \leq 2$ , the threshold  $p_c(N)$  is of the order of  $1/N$  (that is, the value of the natural scale for  $p$ ) and is not observable. From the estimate in Eq. (19) one sees that if  $\gamma > 2$ , it should be close enough to 2 for the extreme resilience of *finite* scale-free networks to failures. One may find a discussion of the resilience of directed equilibrium networks to random damage in [25].

## 6 Open Research Problems

In this chapter we presented the introductory properties of one of the hottest topics in the area of theoretical physics with applications in many branches of science. As can be seen, the field is still in its first stages of development with many open questions. In this final section I will try to motivate the reader to some of the unsolved problems in this very attractive and active field.

There are two levels of problems to be solved: – those related to empirical results and – those related to the analytical understanding of network properties. In the first case many examples come from many branches of science (social networks, Internet, WWW, genoma, proteoma and metabolic reaction, among others). In the second, the challenge is to obtain an exact solution even for the most basic quantities, like clustering, degree distribution, correlations, etc. The known exact results presently correspond to very particular cases.

Some open problems include

- tomography of some networks (WWW, Internet, etc.),
- spectral properties of scale-free networks,
- study of correlations in scale-free networks,
- a general statistical theory for networks (using different ensembles),
- developing statistical tools to analyze social and biological networks, and
- diffusion and first passage time on networks (propagation of information, viruses, etc.).

## 7 Conclusion

This short review presented part of the progress obtained over the last years in the field of networks. Special attention was given to evolving communications networks. In spite of our great progress and increased understanding, it is necessary to admit that most of the discussed models and ideas can only be applied to real networks on a schematic and qualitative level. These simple models are still far from reality and only address particular phenomena in real networks.

The main recent achievements in the theory of networks are related to the study of scale-free networks, and the rapid pace of progress in this field is due to the appearance of exciting networks in our world: the Internet, the WWW, and diverse biological networks. All these studies focus on the structural properties of evolving networks. The two main aspects emerging are the mechanisms associated with the growth (preferential linking), which produces distributions with long tails, and the fact that these networks have different properties compared to the random graphs with Poisson degree distributions. An important aspect is also the robustness and fragility of these networks against attacks and propagation of viruses.

## Acknowledgements

I would like to thank my collaborators S.N. Dorogovtsev, A.N. Samukhin, and A.V. Goltsev, for their help and enthusiasm during the years this topic has been developed in our group. The author was partially supported by the project POC-TI/1999/FIS/33141.

## References

1. L.A. Adamic. The small world Web. In *Proceedings of the 3rd European Conference Research and Advanced Technology for Digital Libraries*, pages 443–452, Paris, France, 1999. ECDL'99.
2. L.A. Adamic and B.A. Huberman. Power-law distribution of the World Wide Web. *Science*, (287):2115a, 2000.
3. L.A. Adamic, R.M. Lukose, A.R. Puniyani, and B.A. Huberman. Search in power-law networks. *Physical Review E*, 64:046135–1–046135–8, 2001.
4. W. Aiello, F. Chung, and L. Lu. A random graph model for massive graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 171–180, 2000.
5. R. Albert and A.-L. Barabási. Topology of evolving networks: Local events and universality. *Phys. Rev. Lett.*, (85):5234, 2000.
6. R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47, 2002.
7. R. Albert, H. Jeong, and A.-L. Barabási. Diameter of the World-Wide Web. *Nature*, 401:130, 1999.
8. R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 2000.
9. U. Alon, M.G. Surette, N. Barkai, and S. Leibler. Robustness in bacterial chemotaxis. *Nature*, (397):381, 1999.
10. J.R. Banavar, A. Maritan, and A. Rinaldo. Size and form in efficient transportation networks. *Nature*, (99):130, 1999.
11. A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, (286):509, 1999.
12. A.-L. Barabási, R. Albert, and H. Jeong. Scale-free characteristics of random networks: The topology of the World Wide Web. *Physica A*, (281):69–77, 2000.

13. A.-L. Barabási, R. Albert, H. Jeong, and G. Bianconi. Power-law distribution of the World Wide Web: response. *Science*, (287):2115a, 2000.
14. A.-L. Barabási, H. Jeong, Z. Néda, Ravasz, A. E., Schubert, and Vicsek T. Evolution of the social network of scientific collaborations. cond-mat/0104162.
15. P. Baran. Introduction to distributed communications networks. Technical Report RM-3420-PR, <http://www.rand.org/publications/RM/baran.list.html>, 1964.
16. N. Barkai and S. Leibler. Robustness in simple biochemical networks. *Nature*, (387):913, 1997.
17. A. Barrat and M. Weigt. On the properties of small-world network models. *Eur. Phys. J. B*, (13):33, 2000.
18. A. Becskei and L. Serrano. Engineering stability in gene networks by autoregulation. *Nature*, (405):590.
19. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th WWW Conference, Brisbane*, page 107, 1998.
20. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure of the Web. In *Proceedings of the 9th WWW Conference*, page 309, 2000.
21. D. Butler. Souped-up search engines. *Nature*, (405):112, 2000.
22. A. Capocci, G. Caldarelli, R. Marchetti, and L. Pietronero. Growing dynamics of Internet providers. cond-mat/0106084.
23. Erez Cohen, R., D. K., Ben-Avraham, and S. Havlin. Resilience of the Internet to random breakdowns. *Phys. Rev. Lett.*, (85):4625, 2000.
24. J. Dean and M.R. Henzinger. Finding related pages in the World Wide Web. *Computer Networks*, (31):1467, 1999.
25. S.N. Dorogovtsev, J.F.F. Mendes, and A.N. Samukhin. Giant strongly connected component of directed networks. *Phys. Rev. E*, (64):025101 (R), 2001.
26. S.N. Dorogovtsev and J.F.F. Mendes. Evolution of networks with aging of sites. *Phys. Rev. E*, (62):1842, 2000.
27. S.N. Dorogovtsev and J.F.F. Mendes. Exactly solvable small-world network. *Europhys. Lett.*, (50):1, 2000.
28. S.N. Dorogovtsev and J.F.F. Mendes. Scaling behavior of developing and decaying networks. *Europhys. Lett.*, (52):33, 2000.
29. S.N. Dorogovtsev and J.F.F. Mendes. Effect of the accelerating growth of communications networks on their structure. *Phys. Rev. E*, (63):025101 (R), 2001.
30. S.N. Dorogovtsev and J.F.F. Mendes. Evolution of networks. *Adv. in Phys.*, 51:1079, 2002.
31. S.N. Dorogovtsev and J.F.F. Mendes. *Evolution of Networks: From Biological Nets to the Internet and WWW*. Oxford University Press, 2003.
32. S.N. Dorogovtsev and J.F.F. Mendes. Comment on “breakdown of the Internet under intentional attack”. *Phys. Rev. Lett.*, 87.
33. S.N. Dorogovtsev, J.F.F. Mendes, and A.N. Samukhin. WWW and Internet models from 1955 till our days and the “popularity” is attractive principle. cond-mat/0009090.
34. S.N. Dorogovtsev, J.F.F. Mendes, and A.N. Samukhin. Structure of growing networks with preferential linking. *Phys. Rev. Lett.*, (85):4633, 2000.
35. S.N. Dorogovtsev, J.F.F. Mendes, and A.N. Samukhin. Generic scale of “scale-free” networks. *Phys. Rev. E*, 062101, 2001.
36. P. Erdős and A. Rényi. On random graphs. *Publications Mathematicae*, (6):290, 1959.
37. P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, (5):17, 1960.
38. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. *Comput. Commun. Rev.*, (29):51, 1999.

39. R. Ferrer, C. Janssen, and R. Solé. The topology of technology graphs: Small world patterns in electronic circuits. Technical report, Working Papers of Santa Fe Institute, 2001.
40. R. Ferrer and R.V. Solé. The small-world of human language. Technical report, Working Papers of Santa Fe Institute, 2001.
41. E. Garfield. *Citation Indexing: Its Theory and Application in Science*. Wiley, New York, 1979.
42. R. Govindan and H. Tangmunarunkit. Heuristics for Internet map discovery. In *Proceedings of the 2000 IEEE INFOCOM Conference*, pages 1371–1380, Tel Aviv, Israel. <http://citeseer.nj.nec.com/govindan00heuristics.html>.
43. B. A. Huberman and L. A. Adamic. Growth dynamics of the World-Wide Web. *Nature*, (401):131, 1999.
44. B. A. Huberman, P.L.T. Pirolli, J.E. Pitkow, and R.J. Lukose. Strong regularities in World Wide Web surfing. *Science*, (280):95, 1998.
45. S. Janson, D.E. Knuth, T. Luczak, and B. Pittel. The birth of the giant component. *Random Structures and Algorithms*, (4):233, 1993.
46. H. Jeong, S.P. Mason, A.-L. Barabási, and Z.N. Oltvai. Lethality and centrality in protein networks. *Nature*, (411):41, 2001.
47. H. Jeong, Z. Néda, and A.-L. Barabási. Measuring preferential attachment for evolving networks. *cond-mat/0104131*.
48. H. Jeong, B. Tombor, R. Albert, Z.N. Oltvai, and A.-L. Barabási. The large-scale organization of metabolic networks. *Nature*, (407):651.
49. J. Kleinberg. The small-world phenomenon: An algorithmic perspective. Technical report, Cornell University Computer Science Department, Technical Report 99-1776, 1999. <http://www.cs.cornell.edu/home/kleinber/swn.ps>.
50. J. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The Web as a graph: measurements, models, and methods. In *Proceedings of the 5th International Conference on Combinatorics and Computing*, pages 1–17, Tokyo, Japan, 1999.
51. J.M. Kleinberg. Navigation in a small world. *Nature*, (406):845, 2000.
52. P.L. Krapivsky and S. Redner. Organization of growing random networks. *Phys. Rev. E*, (63):066123, 2001.
53. P.L. Krapivsky, S. Redner, and F. Leyvraz. Connectivity of growing random networks. *Phys. Rev. Lett.*, (85):4629, 2002.
54. P.L. Krapivsky, G.J. Rodgers, and S. Redner. Degree distributions of growing networks. *Phys. Rev. Lett.*, (86):5401, 2001.
55. R.V. Kulkarni, E. Almaas, and D. Stroud. Exact results and scaling properties of small-world networks. *Phys. Rev. E*, (61):4268, 2000.
56. R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the Web graph. In *Proceedings of the 41th IEEE Symposium on Foundations of Computer Science*, pages 57–65, Redondo Beach, California, 2000.
57. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the Web for emerging cyber-communities. <http://www.almaden.ibm.com/cs/k53/trawling.ps>.
58. S. Lawrence. Context in Web search. *IEEE Data Engineering Bulletin*, (23):25, 2000.
59. S. Lawrence and C.L. Giles. Context and page analysis for improved Web search. *IEEE Internet Computing*, (2):38, 1998.
60. S. Lawrence and C.L. Giles. Searching the World Wide Web. *Science*, (280):98, 1998.
61. S. Lawrence and C.L. Giles. Accessibility of information on the Web. *Nature*, (400):107, 1999.
62. S. Lawrence and C.L. Giles. Searching the Web: General and scientific information access. *IEEE Communications*, (37):116, 1999.

63. F. Liljeros, C.R. Edling, L.A. N. Amaral, H.E. Stanley, and Y. Åberg. The Web of human sexual contacts. *Nature*, (411):907, 2001.
64. A.L. Lloyd and R.M. May. How viruses spread among computers and people. *Science*, (292):1316, 2001.
65. T. Lux and M. Marchesi. Scaling and criticality in a stochastic multi-agent model of a financial market. *Nature*, (397):913, 1999.
66. Molloy M. and B. Reed. A critical point for random graphs with a given degree sequence. *Random Structures and Algorithms*, (6):161, 1995.
67. Molloy M. and B. Reed. The size of the giant component of a random graph with a given degree sequence. *Combinatorics, Probability and Computing*, (7):295, 1998.
68. S. Milgram. The small world problem. *Psychology today*, (2):60, 1967.
69. M.E.J. Newman. The structure of scientific collaboration networks. *Proc. Nat. Acad. Sci. U.S.A.*, (98):404, 2000.
70. M.E.J. Newman. Scientific collaboration networks: I. Network construction and fundamental results. *Phys. Rev. E*, (64):016131, 2001.
71. M.E.J. Newman, S.H. Strogatz, and D.J. Watts. Random graphs with arbitrary degree distribution and their applications. *Phys. Rev. E*, (64):026118, 2001.
72. M.E.J. Newman and D.J. Watts. Scaling and percolation in the small-world network model. *Phys. Rev. E*, (60):7332, 1999.
73. J.-J. Pansiot and D. Grad. On routes and multicast trees in the Internet. *Computer Communications Review*, (28):41, 1998.
74. R. Pastor-Satorras and A. Vespignani. Epidemic spreading in scale-free networks. *Phys. Rev. Lett.*, (86):3200, 2001.
75. Pastor-Satorras R., A. Vázquez, and A. Vespignani. Dynamical and correlation properties of the Internet. cond-mat/0105161.
76. S. Redner. How popular is your paper? an empirical study of citation distribution. *Eur. Phys. J. B*, (4):131, 1998.
77. B. Tadić. Access time of an adaptive random walk on the World-Wide Web. cond-mat/0104029, 2001.
78. C. Tsallis and M.P. de Albuquerque. Are citations of scientific papers a case of nonextensivity? *Eur. Phys. J. B*, (13):4629, 2000.
79. P. Uetz, L. Giot, and G. Cagney et al. A comprehensive analysis of protein-protein interactions in *saccharomyces cerevisiae*. *Nature*, (403):623, 2000.
80. A. Vázquez. Statistics of citation networks. cond-mat/9903433.
81. A. Wagner. The yeast protein interaction network evolves rapidly and contains few redundant duplicate genes. Technical report, Working Papers of Santa Fe Institute, 2001. <http://www.santafe.edu/sfi/publications/Abstracts/01-04-022abs.html>.
82. D.J. Watts and S.H. Strogatz. Collective dynamics of small-world networks. *Nature*, (393):440, 1998.
83. S.-H. Yook, H. Jeong, and A.-L. Barabási. Modeling the Internet's large-scale topology. cond-mat/0107417.



---

# Web Dynamics, Structure, and Page Quality<sup>\*</sup>

Ricardo Baeza-Yates, Carlos Castillo, and Felipe Saint-Jean

Center for Web Research  
Computer Science Department, University of Chile  
Blanco Encalada 2120, Santiago, Chile  
{rbaeza, ccastill, fsaint}@dcc.uchile.cl

**Summary.** In this chapter we aim to study the quantitative measures pertaining to the relationship between the dynamics of the Web, its structure, and the quality of Web pages. Quality is studied using different link-based metrics and considering their relationship with the structure of the Web and the last modification time of a page. We show that, as expected, PageRank is biased against new pages, and we obtain information on how recency is related to Web structure.

## 1 Introduction

The purpose of a Web search engine is to provide an infrastructure that supports relationships between publishers of content and readers. In this context, as the numbers involved are very big (550 million users [2] and more than 3 billion pages (a lower bound that comes from the coverage of popular search engines) in 35 million sites [4] in January 2003), it is critical to provide good measures of quality that allow the user to choose “good” pages. We think that this is the main element that explains Google’s [3] success. However, the notion of what is a “good page” and how this it is related to different Web characteristics is not well understood.

Therefore, in this chapter we address the study of the relationships between the age of a page or a site, the quality of a page, and the structure of the Web. Age is defined as the time since the page was last updated (recency). For Web servers, we use the oldest page in the site as a lower bound on the age of the site.

The specific questions we explore are the following:

- How does the position of a Web site in the structure of the Web depend on the Web site age? Does the quality of a Web page depend on where it is located in the Web structure? We give some experimental data that sheds some light on these issues.

---

<sup>\*</sup> Supported by Millenium Nucleus “Center for Web Research” (P01-029-F), Mideplan, Chile.

- Do link-based ranking schemes provide a fair score to newer pages? We found that the answer is “no” for PageRank [25], which is very important among the ranking functions used by Google [3].

Our study is focused on the Chilean Web, mainly the .cl domain at two different time instants: the first half of 2000, when we collected 670,000 pages in approximately 7,500 Web sites (CL-2000), and the last half of 2001, when we collected 795 thousand pages, corresponding to approximately 21,200 Web sites (CL-2001). This data comes from the TodoCL search engine [5], which specializes on the Chilean Web and is part of a family of vertical search engines built using the Akwan search engine [1].

Most statistical studies about the Web are based either on a “random” subset of the complete Web, or on the contents of some Web sites. In our case, the results are based on a Web collection that represents a large percentage of the Chilean Web. Therefore we believe that our sample is more homogeneous and coherent, because it represents a well-defined cultural and linguistic context.

This chapter is organized as follows. Section 2 presents models of how pages change. Section 3 shows how to measure the rate of change in the Web. Section 4 introduces the main results on Web structure. Section 5 presents several relations among Web structure and age. Section 6 presents the relationship between the quality of Web pages and their age (recency). Finally, we end with some conclusions and future work.

## 2 Models of Page Change

Crawling the World Wide Web resembles, to some extent, the task of an astronomer who watches the sky. What he sees is not a “snapshot” of the instant state of the universe, but rather the result of light traveling from different distances to him. Most of the stars have experienced huge changes or even disappeared by the time their light reaches the earth [7].

In the same way, by the time a Web crawler has finished its crawl, many events can happen on the Web. The following discussion applies to any resource on the public Web space, such as Web pages, documents, images, and audio, but we will focus on text. We are interested in the following events.

### 2.1 Creation

When a page is created, it will not be visible on the public Web space until it is linked, so we assume that at least one page update—adding a link to the new Web page—must occur for a Web page creation to be visible.

### 2.2 Updates

Updates are difficult to characterize. An update can be either *minor* (at the paragraph or sentence level), so the page is still almost the same and references to its content

are still valid; or *major* (all references to its content are not valid anymore). It is customary to consider **any** update as major, as it is difficult to judge automatically if the page's content is semantically the same. Partial change characterization is studied in [24].

### 2.3 Deletions

A page is deleted if it is removed from the public Web, or if all the links to that page are removed. Undetected deletions are more damaging for a search engine's reputation than updates, as they are more evident to the user.

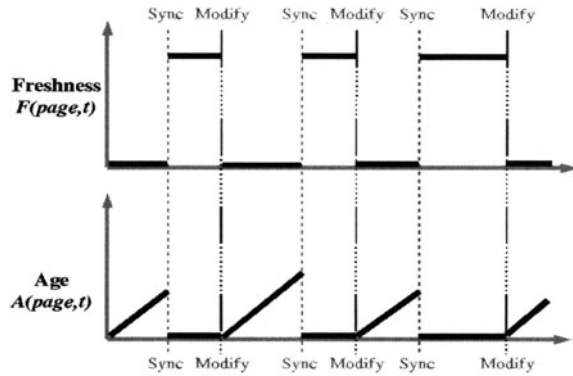
In all cases, there is a cost associated with not detecting an event and having an outdated copy of a resource. The most-used cost functions, defined in [15], are freshness  $F$  and age  $A$ :

$$F(\text{page}_i; t) = \begin{cases} 1, & \text{if } \text{page}_i \text{ is up to date at time } t, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

In this case, an up-to-date page has a value of 1 and an outdated page has a value that degrades linearly with time:

$$A(\text{page}_i; t) = \begin{cases} 0, & \text{if } \text{page}_i \text{ is up to date at time } t, \\ t, & \text{modification time of } \text{page}_i, \text{ otherwise.} \end{cases} \quad (2)$$

The evolution of these two quantities is depicted in Fig. 1.



**Fig. 1.** Evolution of freshness and age with time. Two kinds of events can occur: modification of a Web page in the server, and downloading of the modified page by the crawler (*sync*). When the modified page is downloaded, freshness becomes 1 and age becomes 0

We gathered time information (last-modified date) for each page as informed by the Web servers. We focus on Web page age, that is, the time elapsed after the last modification (recency). As the Web is young, we use months as time units, and our data considers only the three last years as most Web sites are that young. The distribution of pages and sites for CL-2000 with respect to age is given in Fig. 2.

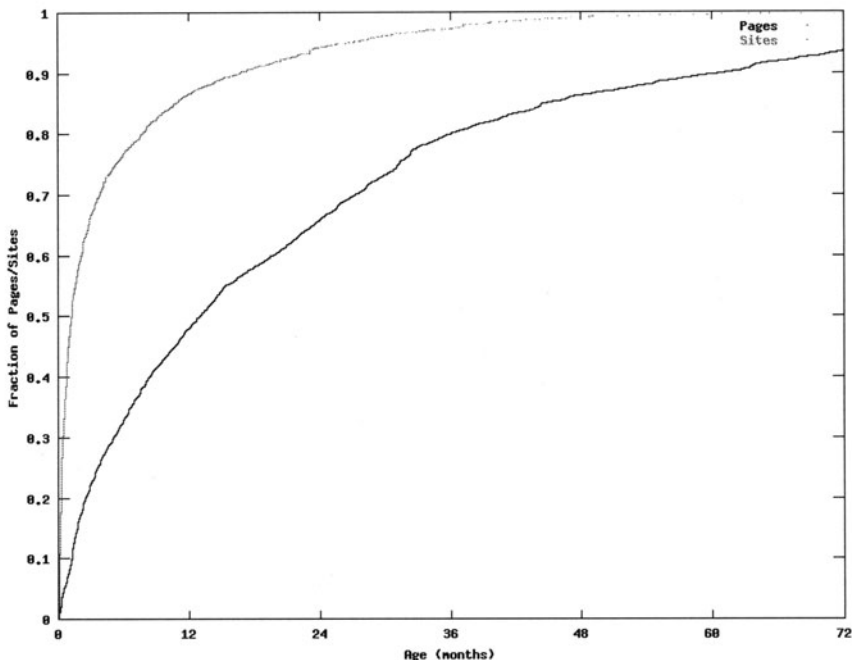


Fig. 2. Cumulative distribution of pages (*bottom*) and sites (*top*) as a function of age for CL-2000

### 3 Measuring and Estimating the Rate of Change

#### 3.1 Sampling

One of the main difficulties involved in any attempt of Web characterization studies is how to obtain a good sample. As there are very few important pages lost in a vast amount of unimportant pages (according to any metric: PageRank, reference count, page size, etc.), just taking a URL at random is not enough. Pages must be sampled based on their importance, and not all pages are known in advance, so a way of estimating the importance of each page based on partial information is needed [20].

There are two approaches for gathering data: using a log of the transactions in the proxy of a large organization or ISP, or using a Web crawler. There are advantages and disadvantages for each method. When monitoring a proxy it is easy to find popular

pages, but the revisit period is impossible to control, as it depends on users; when using a crawler the popularity of pages has to be estimated but the revisit period can be fine-tuned.

### 3.2 Data

The data is obtained by repeated access to a large set of pages during a period of time. Notice that in all cases the results will be only an estimation because they are obtained by **polling** for events (changes), not by the resource **notifying** events.

For each  $page_i$  and each visit we can obtain

- the access timestamp of the page  $visit_i$ ,
- the last-modified timestamp (given by most Web servers; about 80%–90% of the requests in practice)  $modified_i$ , and
- the text page itself, which can be compared to an older copy to detect changes, especially if  $modified_i$  is unknown.

There are other types of data that can be estimated sometimes, especially if the revisiting period is short:

- the creation timestamp of the page,  $created_i$ , and
- the delete timestamp of the page when the page is no longer available,  $deleted_i$ .

### 3.3 Metrics

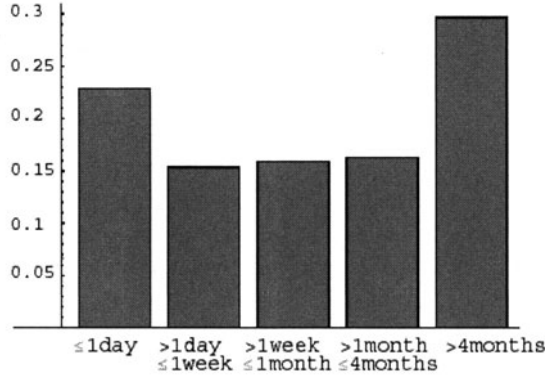
There are different time-related metrics for a Web page, but the most common are

- age,  $visit_i - modified_i$ ,
- lifespan,  $deleted_i - created_i$ ,
- number of modifications during the lifespan,  $changes_i$ , and
- change interval,  $lifespan_i / changes_i$ .

For the entire Web or for a large collection of pages, useful metrics are

- the distribution of change intervals,
- the time it takes for 50% of the Web to change, and
- the average lifespan of pages.

One of the most important metrics is the change interval; Fig. 3 was obtained in a study from 2000 [13]. An estimation of the average change interval is about 4 months.



**Fig. 3.** Fraction of pages with given average interval

### 3.4 Estimating

The probability that a copy of  $page_i$  is up-to-date at time  $t$ ,

$$p_{i,t} = P(F(page_i; t) = 1) \quad (3)$$

decreases with time. Page change can be modeled as a Poisson process [10], so if  $t$  units of time have passed since the last visit:

$$p_{i,t} = e^{-\lambda_i t}. \quad (4)$$

The parameter  $\lambda_i$  characterizes the rate of change of the page  $i$  and can be estimated based on previous observations, especially if the Web server gives the last modification date of the page each time it is visited. The estimation for  $\lambda_i$  [14] is given by

$$\lambda_i \approx \frac{(X_i - 1) - \frac{X_i}{N_i \log(1 - X_i/N_i)}}{S_i T_i}, \quad (5)$$

where

- $N_i$  is the number of visits to  $page_i$ .
- $S_i$  is the time since the first visit to  $page_i$ .
- $X_i$  is the number of times the server has informed that the page has changed.
- $T_i$  is the total time with no modification, according to the server, summed for all the visits.

If the server does not give the last-modified time, we can still check for modifications by comparing the downloaded copies at two different times, so  $X_i$  will now be the number of times a modification is detected. The estimation for the parameter in this case is

$$\lambda_i \approx \frac{-N_i \log(1 - X_i/N_i)}{S_i}. \quad (6)$$

There have been many large-scale studies about Web dynamics in 1997 [18], 1999 [21], 2000 [10, 11], and a recent study in 2003 [19]. An important motivation for them is determining an optimal scheduling policy for a Web crawler, as in the works by Cho [13, 16, 14, 15] and Coffman [17]. In this area, there are some proposals for changing from a polling scheme to a notify scheme [26], in which servers cooperate with crawlers [9].

## 4 Structure of the Web

The most complete study of the Web structure [12] focuses on the connectivity of Web pages. This study starts with a large-scale Web graph and then identifies a single large strongly connected component (MAIN). All the structures in the graph are related to MAIN, as explained later.

A page can describe several documents and one document can be stored in several pages, so we decided to study the structure of how Web sites were connected, as Web sites are closer to real logical units. Not surprisingly, we found in [6] that the structure in the .cl (Chile) domain at the Web site level was similar to the global Web—another example of the autosimilarity of the Web, which gives a scale invariant—and hence we use the same notation of [12]. The components are

- MAIN: sites that are in the strongly connected component of the connectivity graph of sites (that is, we can navigate from any site to any other site in the same component);
- IN: sites that can reach MAIN but cannot be reached from MAIN;
- OUT: sites that can be reached from MAIN, but there is no path to go back to MAIN;
- other sites that can be reached from IN (T.IN, where T is an abbreviation of tentacle), sites in paths between IN and OUT (TUNNEL), sites that only reach OUT (T.OUT), and unconnected sites (ISLANDS).

In [6] we analyzed CL-2000 and we extended this notation by dividing the MAIN component into four parts:

1. MAIN–MAIN, which are sites that can be reached directly from the IN component and can reach directly the OUT component;
2. MAIN–IN, which are sites that can be reached directly from the IN component but are not in MAIN–MAIN;
3. MAIN–OUT, which are sites that can reach directly the OUT component, but are not in MAIN–MAIN;
4. MAIN–NORM, which are sites not belonging to the previously defined subcomponents.

Fig. 4 shows all these components.

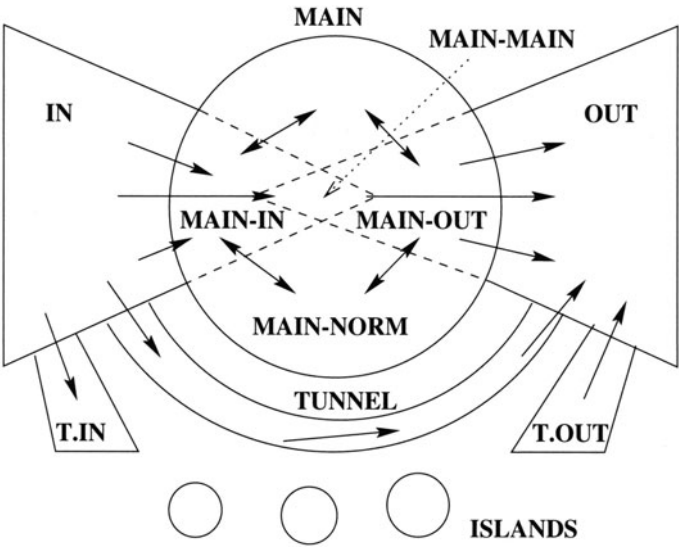


Fig. 4. Structure of the Web

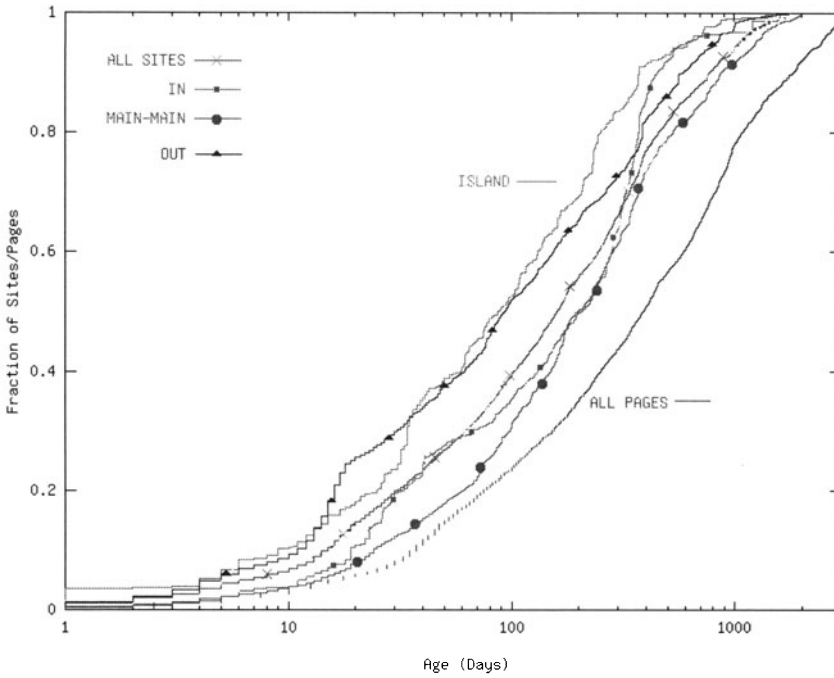
An important observation is that in random graphs the size of the strongly connected component tends to be much larger, that is, it contains over 90% of the nodes. This is not the case for the Web. This can be explained because the Web is not a random graph; it grows in a way that is compatible with models of preferential attachment, in which a highly linked page tends to attract even more links [22]. However, these models do not consider that part of the Web also disappears. More on generative models for the Web can be found in the chapter by Mendes.

## 5 Web Structure and Age

An initial motivation is to find if the IN and OUT components were related to Web dynamics or just due to bad Web site design. In fact, Web sites in IN could be considered as new sites that are not linked because of causality reasons. Similarly, OUT sites could be old sites that have not been updated. Figure 5 plots the cumulative distribution of the oldest page in each site for Set 1 in each component of the Web structure versus date in a logarithmic scale (these curves have the same shape as the ones in [12] for pages). The central part is a line and represents the typical power laws that appear in many Web measures.

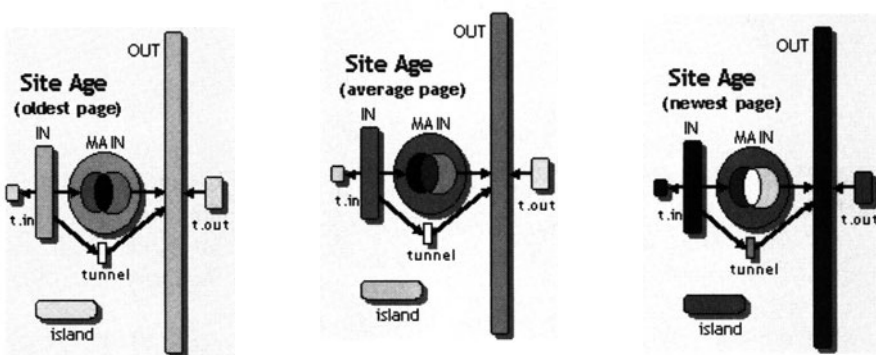
Figure 6 shows the relation between the macrostructure of the Web using the number of Web sites in each component to represent the area of each part of the diagram for CL-2000. The gray scale represents Web site age, such that a darker tone represents older Web sites. We consider three ages: the oldest page, which is a lower bound to the Web site age, the average age, which can be considered as the freshness of a site, and the newest page, which is a measure of update frequency on





**Fig. 5.** Web site age in the different components and Web page age (*rightmost curve*)

a site. The diagrams in Fig. 6 show that the oldest sites are in MAIN–MAIN, while the sites that are fresher on average are in MAIN–IN and MAIN–MAIN. Finally, the last diagram at the right shows that the update frequency is high in MAIN–MAIN and MAIN–OUT, while sites in IN and OUT are updated less frequently.



**Fig. 6.** Visualization of Web structure and Web site age

We also obtain some confirmation of our expectations. Newer sites are in the ISLANDS component (and therefore they are not linked, yet). The oldest sites are

in MAIN, in particular MAIN–MAIN, so the kernel of the Web comes mostly from the past. What is not obvious, is that, on average, sites in OUT are also newer than the sites in other components. We observed that some of these Web sites are from e-commerce sites whose policy is to not link to other Web sites.

Finally, IN shows two different parts: there is a group of new sites, but the majority are old sites. Hence, a large fraction of IN are sites that never became popular.

In Table 1 we give the numerical data for the average of Web site age (using the oldest page) as well as the overall Web quality (sum for all the sites) in each component of the macrostructure of the Web, as well as the percentage change among both data sets in more than a year. Although CL-2000 did not include all the ISLANDS at that time (we estimate that CL-2000 was 70% of the sites), we can compare the core. The core has the smaller percentage but it is larger as CL-2001 triples the number of sites of CL-2000. The OUT component has also increased, which may imply a degradation of some part of the Web. Inside the core, MAIN–MAIN has increased at the expense of MAIN–NORM. Overall, CL-2001 represents a Web that is much more connected than CL-2000.

Component	size (%CL-2000)	size (%CL-2001)	age (days)
MAIN	23	9	429
IN	15	6	295
OUT	45	20	288
TUNNEL	1	0	329
T.IN	3	3	256
T.OUT	9	2	293
ISLANDS	4	60	273
MAIN-MAIN	2	3	488
MAIN-OUT	6	2	381
MAIN-IN	3	1	420
MAIN-NORM	12	2	395

**Table 1.** Distribution and age CL-2001 in the different components of the macrostructure of the Chilean Web

Several observations can be made from Table 1. First, sites in MAIN has the higher PageRanks, and inside it, MAIN–MAIN is the subcomponent with the highest PageRank. In a similar way, MAIN–MAIN has the largest authority. This makes MAIN–MAIN a very important segment of the Web. Notice that IN has the higher hub, which is natural because sites in MAIN have the higher authority. ISLANDS have a low score in all cases.

Studying age, sites in MAIN are the oldest, and inside it, sites in MAIN–MAIN are the oldest. As MAIN–MAIN also has good quality, it seems that older sites have the best content. This may be true when evaluating the quality of the content, but the value of the content, we believe in many cases, could be higher for newer pages, as we need to add novelty to the content.

Table 2 indicates the percentage of sites coming from CL-2000 in each part of the structure for CL-2001. Note that some are new sites (NEW) and that other sites from CL-2000 have disappeared (GONE). Figure 7 shows this change graphically using relative percentage changes in each component and absolute percentage changes in the arrows. Each column has new sites and the distribution of the sources of old sites. The main flows are from MAIN, IN, and OUT to ISLANDS; or from MAIN to OUT (probably sites that become outdated), and sites that disappear in OUT and ISLANDS (probably new sites that were not successful). On the other hand, it is interesting to notice the stability of the MAIN component. At the same time, all these changes show that the Web is very unstable as a whole. Therefore there is a strong relation between the macrostructure of the Web and age and quality characteristics. This implies that the macrostructure is a valid partition of Web sites regarding these characteristics.

2000 - 2001	MAIN	IN	OUT	ISLANDS	GONE
MAIN	36.36	5.31	27.46	11.57	19.30
IN	5.19	15.71	11.85	37.15	30.09
OUT	8.12	1.62	31.21	31.21	27.83
ISLANDS	3.31	2.58	22.84	39.23	32.04
NEW	5.20	6.30	14.40	74.10	
Rest	3.79	11.76	29.41	1.26	53.78

**Table 2.** Relative percentage of sites coming from different parts of the structure, including new sites and sites that have disappeared among CL-2000 and CL-2001

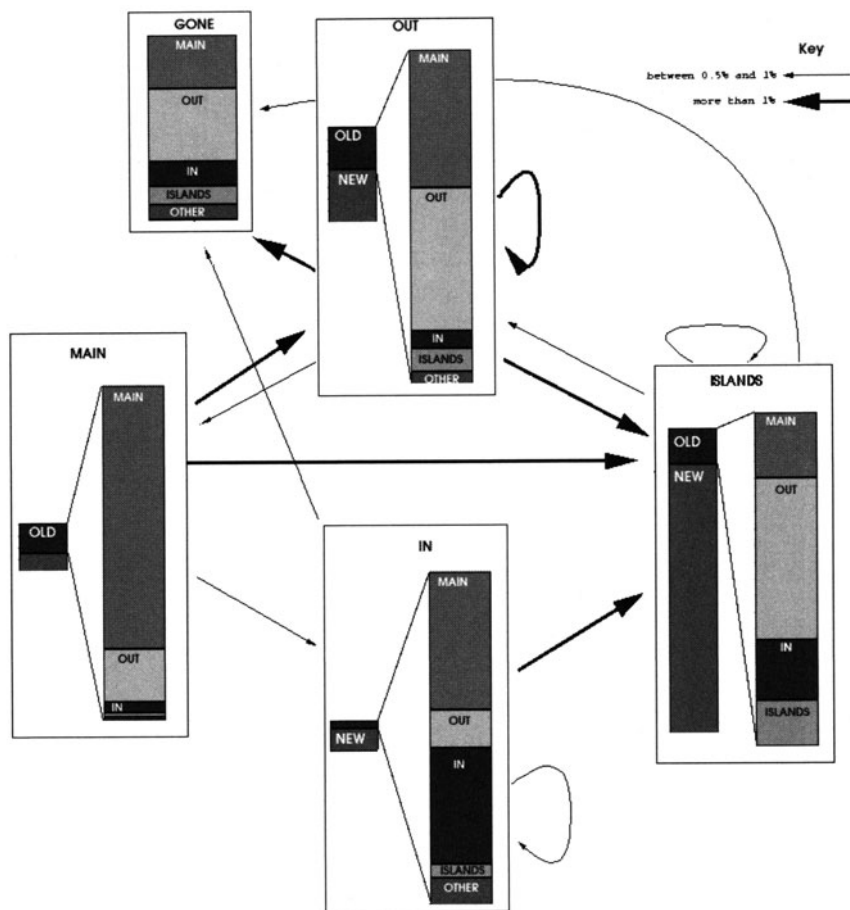
## 6 Link-Based Ranking and Age

The two main link based ranking algorithms known in the literature are PageRank [25] and the hub and authority measures of the *Hyperlink-Induced Topic Search* (HITS) algorithm [23]. PageRank is based on the probability of a random surfer to be visiting a page. This probability is modeled with two actions: the chance of the surfer to get bored and jump randomly to any page in the Web (with uniform probability), or to choose randomly one of the links in the page. This defines a Markov chain, that converges to a permanent state, where the probabilities are defined as follows:

$$PR_i = \frac{q}{T} + (1 - q) \sum_{j=1, j \neq i}^k \frac{PR_{m_j}}{L_{m_j}},$$

where  $T$  is the total number of Web pages,  $q$  is the probability of getting bored (typically 0.15),  $m_j$  with  $j \in (1, \dots, k)$  are the pages that point to page  $i$ , and  $L_j$  is the number of outgoing links in page  $j$ .

The hub and authority scores are complementary functions. A page will have a high hub rank if it points to good content pages. Similarly, a page will have a high



**Fig. 7.** Site flow among the Web structure from 2000 to 2001 (the *left column* in each box indicates new and old sites, the *right column* indicates from where the old sites come)

authority rank if it is referred to by pages with good links. In this way the authority of a page is defined as the sum of the hub ranks of the pages that point to it, and the hub rank of a page is the sum of the authority of the pages it points to.

Hubs and authorities were defined for subsets of the Web graph induced by a textual query, usually called “dynamic ranking”. In this study, we use them in terms of the whole graph (“static ranking”). Table 3 shows the average page static ranking in each component of the structure.

When considering the rank of a Web site, we use the sum of all the ranks of the pages in the site, which is equivalent to the probability of being in any page of the site in the case of PageRank [6]. Using this is fairer than using the best page or the average of all pages of a site.

Component	PageRank	Hub	Authority
MAIN	2e-04	53e-04	9e-04
IN	0.8e-04	542e-04	1e-07
OUT	0.6e-04	1e-07	0.1e-04
TUNNEL	0.2e-04	1e-07	0e-07
T.IN	0.3e-04	0e-07	0e-07
T.OUT	0.4e-04	0e-07	0e-07
ISLANDS	0.1e-04	0e-07	0e-07
MAIN-MAIN	3e-04	144e-04	25e-04
MAIN-OUT	1e-04	1e-04	4e-07
MAIN-IN	1e-04	0e-07	98e-07
MAIN-NORM	0.8e-04	0e-07	2e-07

**Table 3.** Page quality for CL-2001 in the different components of the macrostructure of the Chilean Web

In [6] we gave qualitative data that showed that link-based ranking algorithms had bad correlation and that PageRank was biased against new pages. Here we present quantitative data supporting those observations.

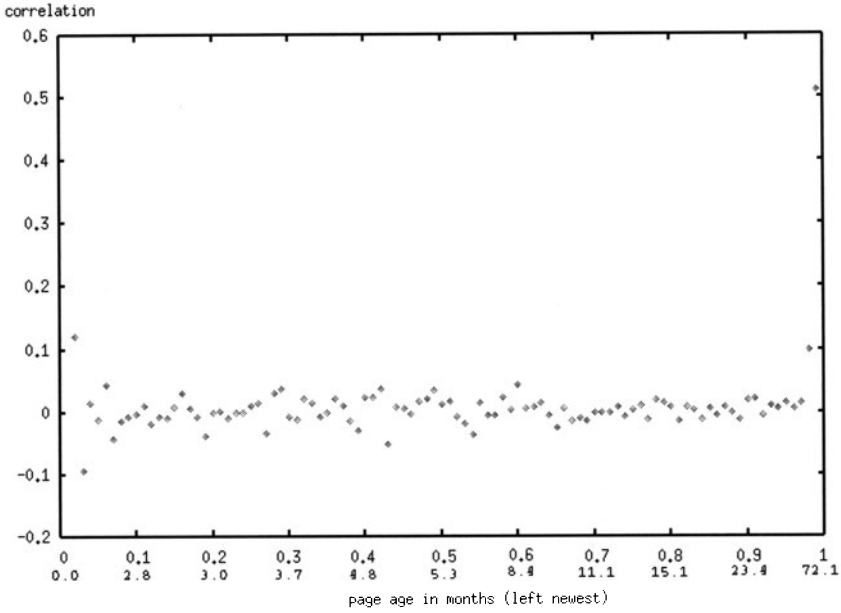
Web pages sorted by recency were divided into 100 group segments of the same weight (that is, each segment has the same number of pages), obtaining a time division that is not uniform. Then we calculated the standard correlation (which is defined as  $\hat{\rho}(x, y) \equiv \frac{cov(x, y)}{\hat{\sigma}_x \hat{\sigma}_y}$  where  $x$  and  $y$  are two random variables,  $cov$  is the covariance, and  $\sigma$  is the standard deviation) of each pair of average rank values. Three graphs were obtained: Fig. 8 which shows the correlation between PageRank and authority, Fig. 9, which shows the correlation among PageRank and hub, and Fig. 10, which shows the correlation of authorities and hubs. Notice that the horizontal axis has two scales: the fraction of groups (top) and the recency in months (bottom).

The low correlation between PageRank and authority is surprising because both ranks are based on incoming links. This means that PageRank and authority are different for almost every age percentile except the one corresponding to the older and newer pages, which have PageRank and authority rank very close to the minimum.

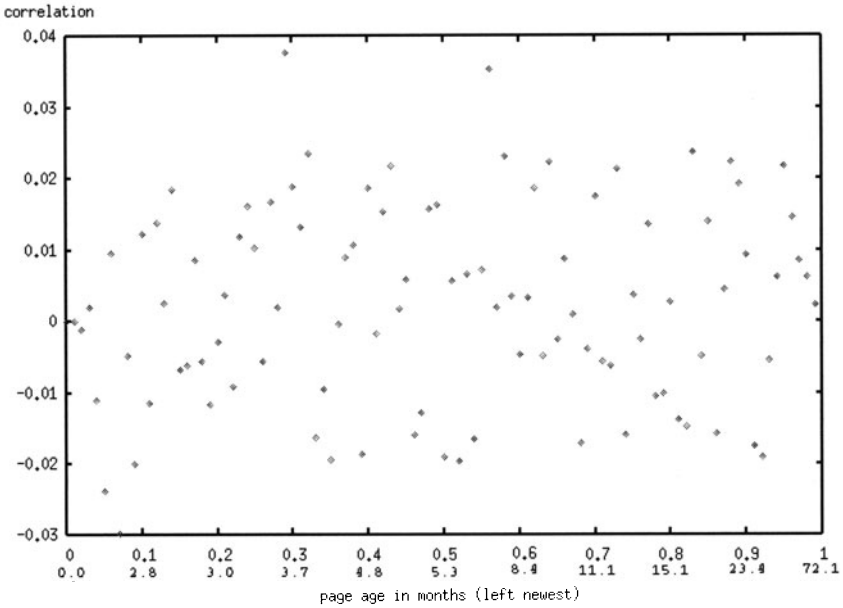
Notice the correlation between hub/authority, which is relatively low but with higher value for pages about 8 months old. New pages and old pages have a lower correlation. Also notice that hub and authority are not biased with time.

It is intuitive that new sites will have low PageRank because Web masters of other sites take time to know the site and refer to it in their sites. This is true also for other ranking schema based on incoming links. We show that this intuition is correct in Fig. 11, where PageRank is plotted against percentiles of page age. As can be seen, the newest pages have a very low PageRank, similar to very old pages. The peak of PageRank is in three months old pages.

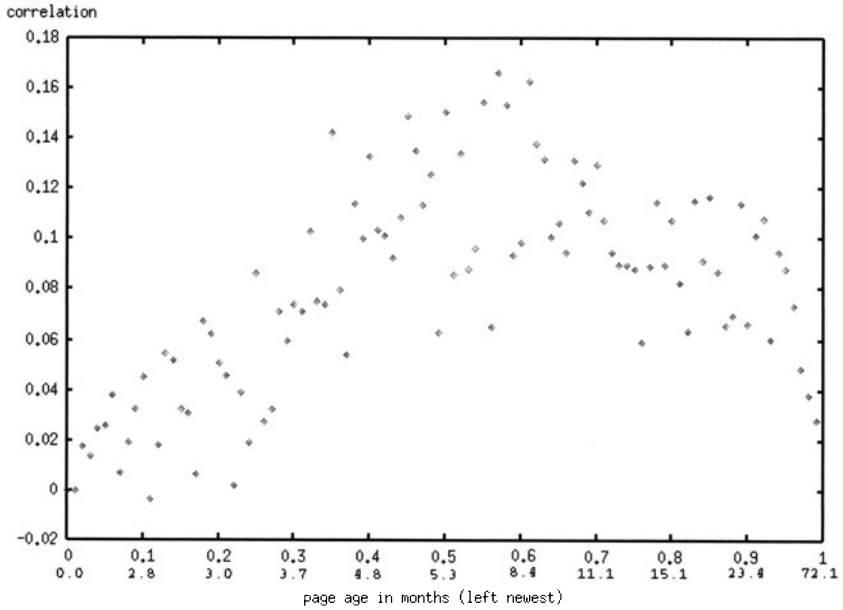
In a dynamic environment such as the Web, new pages have a high value so a ranking algorithm should take an updated or new page as a valuable one. Pages with high PageRank are usually good pages, but the opposite is not necessarily true (good precision does not imply good recall). So the answer is incomplete, and a missing



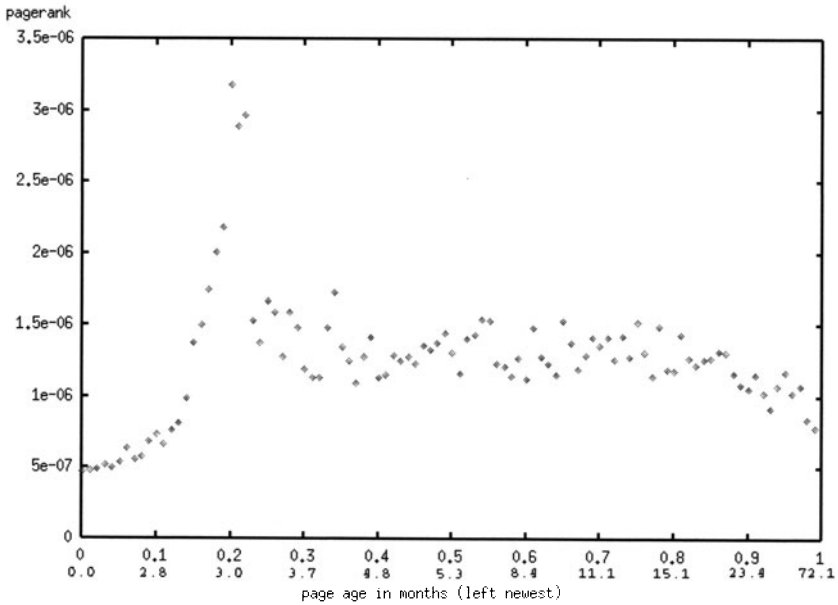
**Fig. 8.** Correlation between PageRank and authority with age



**Fig. 9.** Correlation between PageRank and hub with age



**Fig. 10.** Correlation between hubs and authorities with age



**Fig. 11.** PageRank as a function of page age

part of it is in new pages. An age-based PageRank based on these results is presented in [8], using the following expression:

$$PR_i = \frac{q}{T} + (1 - q) f(\text{age}_i) \sum_{j=1, j \neq i}^k \frac{PR_{m_j}}{L_{m_j}}, \quad (7)$$

where  $f(\text{age}_i)$  is the last-modified date of the page. In [8] we use

$$f(\text{age}) = (1 + A * e^{-B * \text{age}})$$

with positive values of  $A$  and  $B$  to obtain a PageRank scheme biased to new pages. The idea is that each time a page is changed, implies a reaffirmation of the links on that page. Hence if the page is new, the value of the link is  $1 + A$ . If the page is never updated, the value of the link tends to 1.

## 7 Conclusions

In this chapter we showed several relations between the macrostructure of the Web, page and site age, and quality of pages and sites. There is plenty to do to mine the presented data, and this is just the beginning of this kind of Web mining. For example, it is usually assumed that PageRank never decreases, as incoming links should only increase. However, as shown previously, part of the Web disappears, so incoming links to a page may decrease. Including Web death in generative models and analysis of ranking measures are interesting open problems. Further related work includes how to evaluate the real goodness of a Web page link-based ranking and the analysis of search engine logs to study user behaviour with respect to time.

## References

1. Akwan search engine: Main page. <http://www.akwan.com>, 1999.
2. Nua internet - how many online. [www.nua.ie/surveys/how\\_many\\_online/](http://www.nua.ie/surveys/how_many_online/), 2001.
3. Google search engine. [www.google.com/](http://www.google.com/), 2002.
4. Netcraft Web server survey. [www.netcraft.com/survey/](http://www.netcraft.com/survey/), 2002.
5. Todo.cl - todo chile en internet. [www.todo.cl/](http://www.todo.cl/), 2002.
6. R. Baeza-Yates and C. Castillo. Relating Web characteristics with link based Web page ranking. In *String Processing and Information Retrieval*. IEEE Cs. Press, 2001.
7. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley and ACM, Harlow, UK, 1999.
8. R. Baeza-Yates, F. Saint-Jean, and C. Castillo. Web structure, dynamics and page quality. In *Proceedings of the 9th Symposium on String Processing and Information Retrieval (SPIRE'2002)*, pages 117–130. Springer Lecture Notes in Computer Science, 2002.
9. O. Brandman, J. Cho, H. Garcia-Molina, and N. Shivakumar. Crawler-friendly Web servers. In *Proceedings of the Workshop on Performance and Architecture of Web Servers (PAWS)*, Santa Clara, California, 2000.



10. B.E. Brewington and G. Cybenko. How dynamic is the Web? In *World Wide Web Conference*, 2000.
11. B.E. Brewington and G. Cybenko. Keeping up with the changing Web. *Computer*, 33(5):52–58, 2000.
12. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, and A. Tomkins. Graph structure in the Web: Experiments and models. In *9th World Wide Web Conference*, 2000.
13. J. Cho. The evolution of the Web and implications for an incremental crawler. In *Proceedings of International Conference on Very Large Data Bases*, pages 200–209, Cairo, Egypt, 2000.
14. J. Cho and H. Garcia-Molina. Estimating frequency of change. In *Technical Report*, 2000.
15. J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *ACM International Conference on Management of Data (SIGMOD)*, pages 117–128, Dallas, Texas, 2000.
16. J. Cho and A. Ntoulas. Effective change detection using sampling. In *Proceedings of the 28th Very Large Databases Conference*, pages 514–525, 2002.
17. E.G. Coffman Jr., Z. Liu, and R.R. Weber. Optimal robot scheduling for Web search engines. Technical Report RR-3317, INRIA, 1997.
18. F. Douglass, A. Feldmann, B. Krishnamurthy, and J.C. Mogul. Rate of change and other metrics: a live study of the World Wide Web. In *USENIX Symposium on Internet Technologies and Systems*, 1997.
19. D. Fetterly, M. Manasse, M. Najork, and J.L. Wiener. A large-scale study of the evolution of Web pages. In *World Wide Web Conference*, pages 669–678, 2003.
20. M. Henzinger, A. Heydon, M. Mizenmacher, and M. Najork. On near-uniform URL sampling. In *World Wide Web Conference*, 2000.
21. B. Huberman and L. Adamic. Evolutionary dynamics of the World Wide Web. Technical report, Xerox Palo Alto Research Center, 1999.
22. B. Huberman and L. Adamic. Growth dynamics of the World Wide Web. *Nature*, 401:131, 1999.
23. J. Kleinberg. Authoritative sources in a hyperlinked environment. In *9th Symposium on discrete algorithms*, pages 668–677, San Francisco, 1998.
24. L. Lim, M. Wang, S. Padmanabhan, J.S. Vitter, and R. Agarwal. Characterizing Web document change. *Lecture Notes in Computer Science*, 2118:133–144, 2001.
25. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the Web. Working paper, Department of Computer Science, Stanford University, 1998.
26. S. Reddy and M. Fisher. Event notification protocol. Technical report, WEBDAV Working Group Internet Draft, 1998.

## **Part II**

---

### **Searching and Navigating the Web**

---

## Introduction

This section of the book focuses on the dynamics of the interface between the Web and its users through search and navigation. Search engines have become the gatekeepers of the Web and without them much of the Web's information would be inaccessible. Web search engines are faced with a massive scalability problem resulting in an ongoing effort to keep their indexes up-to-date, and so the focus of this section, looking into the technologies underlying the fundamental information seeking strategies of search and navigation, will continue to be very topical.

Chapter 6 focuses on navigation (or “surfing”) of the Web – the activity of clicking on links and browsing Web pages. Coupled with the use of search engines, this is the predominant means by which Web users find their way through the Web. In contrast to search engine technology, which has been steadily improving, there is still a lack of advanced navigation tools to help users orient themselves during their surfing activity. An important problem that is addressed in this chapter is that of users “getting lost in hyperspace” while they are surfing, due to losing the context in which they are browsing, and as a result being unsure how to proceed in order to satisfy their information needs.

Levene and Wheeldon survey previous work, presenting an historical perspective of navigation in hypertext and detailing various navigation tools that have been developed to assist users in orienting themselves within the Web topology and for finding their way to achieve their surfing goals. A brief introduction to more recent work on navigating the mobile Web is presented, and some discussion is given on how viewing the Web as a social network may lead to improved navigation techniques.

The authors present the details of some algorithms that underpin a Web navigation system they have been developing. They introduce a metric, called the potential gain, which assigns scores to Web pages according to their ability to act as “good” starting points for user navigation. In addition, the Best Trail algorithm, which is a probabilistic best-first algorithm that dynamically finds trails that are relevant to a user query, is presented. Finally, a model for analysing the record of trails that emerge from user navigation over time is described. One of the open problems mentioned is that of developing a coherent framework for evaluating user trails within the Web.

Chapter 7 focuses on the problems of developing crawlers that surf the Web and automatically download Web pages for analysis and indexing purposes. Because of the size of the Web and the importance of coverage for global search engines, their crawlers must currently be able to reach about a hundred million Web pages a day. As a result, optimising their performance and strategy for deciding the order in which to visit Web pages is crucial.

Pant et al. present a comprehensive overview of the various components of a crawler, pointing out that crawling can be viewed as a graph search problem. The simplest type of crawler uses a breadth-first strategy that exhaustively crawls the Web as is required by Web search engines in order to build their indexes. As opposed to blindly crawling the Web, preferential crawlers are selective in the Web pages they fetch. Of particular interest are a subclass of preferential crawlers, called focused crawlers, that retrieve only pages within a specified topic. Many of the focused crawling algorithms are based on a best-first heuristic that, for example, first downloads the Web page that best matches an input query.

The authors also discuss the evaluation of crawlers and mention the difficulty with respect to focused crawling because of the large scale of the operation and the inability to determine, in an absolute sense, whether a page is relevant. Metrics analogous to precision and recall from information retrieval are suggested and some experiments that have been carried out comparing breadth-first and best-first crawlers are discussed. One of the open problems mentioned is how to utilise search engines to help a crawler focus on a topic.

Chapter 8 looks at the combination of link and content analysis. The empirical observation that Web pages tend to be linked to other Web pages with similar content is the basis for this combination and is explored by the authors.

Richardson and Domingos survey the two well-known algorithms for link analysis, Kleinberg's HITS algorithm and Google's PageRank algorithm, which were introduced in Chapter 3 in the context of the identification of Web communities. They point out that HITS, which is query specific, suffers from topic drift, i.e. the problem of including Web pages that are peripheral to the topic of the query, and that it also suffers from being too time consuming for current search engines to compute in realtime. On the other hand, PageRank is query independent and so can be computed offline, but when combined with query results online it too leads to topic drift.

Their solution is to compute the PageRank individually for each query, therefore concentrating only on pages that are relevant to the query and thus avoiding topic drift. They propose to precompute the query dependent PageRank for individual query terms and then combine these when the user issues a query. The authors present empirical results demonstrating the scalability of their approach. One of the open problems mentioned is that of incorporating additional measures of the quality of a Web page into the search engine's scoring function, apart from content of pages and link analysis.

Chapter 9 introduces measures for evaluating search engines based on the dynamic nature of the Web. Search engines are faced with the impossible task of keeping their indexes fresh in an environment that is undergoing continuous change. In order to

cope with this change, search engines have to crawl the Web on a regular basis and also stabilise their information retrieval algorithms.

Bar-Ilan surveys previous work on search engine evaluation, where measures such as precision and recall do not take into account the dynamic nature of the Web. She also surveys research done on monitoring the rate of change in Web pages and on experiments carried out on fixed sets of Web pages monitoring their change over time. Other experiments show that for fixed queries the results from search engines fluctuate over time. Several reasons for these fluctuations are mentioned, some related to the freshness of the search engine indexes and others to the search engine's algorithmics.

The author proposes a new set of evaluation measures pertaining to the freshness of the search engine's indexes, i.e. how much does it differ from the current state of the Web, and its stability over time, that is, how much do its results fluctuate over time. One of the open problems mentioned is to look at the effect of commercial concerns such as paid inclusion and ranking on the performance of a search engine.

---

# Navigating the World Wide Web

Mark Levene and Richard Wheeldon

School of Computer Science and Information Systems  
Birkbeck University of London  
Malet Street, London, WC1E 7HX, U.K.,  
{mark,richard}@dcs.bbk.ac.uk

**Summary.** Navigation (colloquially known as “surfing”) is the activity of following links and browsing Web pages. This is a time-intensive activity engaging all Web users seeking information. We often get “lost in hyperspace” when we lose the context in which we are browsing, giving rise to the infamous *navigation problem*. So, in this age of information overload we need navigational assistance to help us find our way through the tangled Web of pages and links. Search engines partially solve this problem by locating relevant documents and finding “good” starting points for navigation, but more navigational assistance is needed to guide users through and between Web sites.

We present a model for navigation which has enabled us to develop several tools and algorithms for helping users with their navigation difficulties. Our point of view is that to help understand how users navigate the Web topology we can attach probabilities to links giving rise to a probabilistic automaton, which can also be viewed as a Markov chain. These probabilities have two interpretations, namely they can denote the proportion of times a user (or a group of users) followed a link, or alternatively they can denote the relevance (or expected utility) of following a link.

We present a new metric for measuring the navigational potential of a Web page, called the *potential gain*. This metric is used to find “good” starting point for an algorithm we describe in detail, called the *Best Trail* algorithm, which semiautomates Web navigation by deriving relevant trails given a user query. We also present techniques we have developed in the area of Web usage mining, detailing our algorithms for analysing records of trails that emerge from either an individual user or a group of users through navigation within the Web graph over a period a time.

We also give historical and current overviews of attempts to address the navigation problem, and review the various navigation tools available to the Web surfer. Finally, we give a brief introduction to navigating within the mobile Web and discuss new navigation techniques that have arisen from viewing the Web as a social network.

## 1 Introduction

We are living in an era of information overload, where finding relevant content is becoming increasingly difficult. The World Wide Web (the Web) collates a massive

amount of online information of varying quality, some of which can found through search engines and some which can only be traced through intensive browsing of Web pages coupled with navigation via link following. As an indication of the massive volume of the Web, a recent estimate of its size given by Murray of Cyveillance during 2000 [49] was 2.1 billion pages, and more recently towards the end of 2001, Google reported that their index contained over 3 billion Web documents; see [www.google.com/press/pressrel/3billion.html](http://www.google.com/press/pressrel/3billion.html). We expect that the actual number of Web pages is much higher than 3 billion, as each search engine covers only a fraction of the totality of accessible Web pages [36]. Moreover, this estimate does not include *deep Web* data contained in databases which are not accessible to search engines [5].

A user seeking information on the Web will normally iterate through the following steps:

1. *Query*: the user submits a query to a search engine specifying his or her goal; normally a query consists of one or more input keywords.
2. *Selection*: the user selects one the returned links from the ranked list of pages presented by the search engine, and browses that Web page displayed as a result of clicking the link.
3. *Navigation* (colloquially known as surfing): the user initiates a navigation session, which involves following links highlighted by link text and browsing the Web pages displayed.
4. *Query modification*: a navigation session may be interrupted for the purpose of query modification, when the user decides to reformulate the original query and resubmit it to the search engine. In this case the user *returns* to step 1.

In other cases the user may go directly to a home page of a Web site or some other starting point, and start navigation by iterating through steps 2 and 3.

Behind each query to a global search engine there is an information need, which according to Broder [11] can be classified into three types:

1. Informational: when the intent is to acquire some information presumed to be present in one or more Web pages.
2. Navigational: when the intent is to find a particular site from which the user can start surfing.
3. Transactional: when the intent is to perform some activity which is mediated by a Web site, for example, online shopping.

Depending on the specification of the query and the quality of the search engine, the user issuing an informational or transactional query may satisfy his or her information need with minimal navigation and query modification. For example, if the user is interested in a particular paper by a given author, he or she may find the paper directly through an informational query such as “bush as we may think”. As an example of a transactional query, the keywords “bargains online bookshop”, would point the user towards online bookshops where further interaction with the user will take place within the online bookshop rather than with the global search engine. In this case the user will probably interact with a local search engine and may have to navigate within

the Web site to find the information needed. As an example of a navigational query, the user may wish find Tim Berners-Lee's keynote address in the WWW2002 conference by issuing the query, "www2002 conference", and then following the appropriate links to the desired information. In this case the effort the user needs to expend to find the keynote address depends on two factors: on the navigational assistance provided within the WWW2002 Web site, and on the navigational competence of the user in picking up the available navigation cues within the WWW2002 Web site. Although Broder's taxonomy was devised with global search in mind, it is also relevant for search within medium to large Web sites, with the user's starting point often being the home page of the site.

Here we will limit ourselves to step 3 of the information-seeking process, i.e. the navigation step, which is not directly supported by search engines. Although search engines can provide a user with "good" Web pages for starting a navigation session, once the user is surfing the search engine provides *no* additional support to assist the user in realising his or her goal. If the navigation process is unsuccessful the user may opt to modify the query through step 4 or go back to a previous page and choose a different link to follow.

It may be argued that it is not within the scope of search engines to provide navigational assistance and that first, the browser should supply some navigation tools, and secondly, Web sites should aid users navigating within them. To some degree this is a valid argument (see Sect. 5), but we take the view that search engines can give help by providing contextual information in the form of trails (see Sect. 7). The notion of a *trail*, inspired by Bush's vision [14], is defined as a sequence of links which may be followed by the user at step 3 of the information-seeking process. Thus navigation is the process enacted when following a trail of information, where the value of the trail as a whole is, in general, greater than the individual values of pages on the trail. We believe that trails should be first-class objects in the Web, in addition to pages and links, which are considered to be the basic building blocks of any hypertext [51].

Providing navigational support in the Web, and, in general, in any hypertext, is important due to the infamous *navigation problem*, whereby users become "lost in hyperspace" [51], meaning that they become disoriented in terms of

- where they are relative to prominent pages such as the home page,
- what they should do next in order to achieve their goal, and
- how they can return to a previously browsed page.

In other words, Web surfers or, more generally, hypertext readers may lose the context in which they are browsing and need assistance in finding their way. In this critical review we will concentrate on ways of tackling the navigation problem, mainly within the Web, which is the definitive existing global hypertext.

Our starting point will be the presentation of a formal model of hypertext and the Web, which will provide us with a useful reference point (see Sect. 2). We will then review the field from an historical perspective starting from Bush's seminal



work on the *memex* [14], through Nelson's *Xanadu* project and his vision of a global hypertext [50], culminating in the current day Web invented by Berners-Lee, the current director of the World Wide Web Consortium, which is most influential in directing the evolution of the Web [6]. Following that we will review the navigation problem from a hypertext perspective, leading to recent Web-specific proposals for tackling the problem (see Sect. 4). We will not ignore recent developments relating to the *mobile Web* [62], where access to the Web is becoming pervasive through a wide variety of input modalities and computing devices, such as voice via mobile phones and pens via handheld PCs (see Sect. 9). In Sect. 10 we will briefly review recent work, which shows that viewing the Web as a social network leads to new techniques for tackling the navigation problem. Finally, in Sect. 11 we list some open problems that warrant further investigation.

We do not attempt to review all recent work in this area as the number of publications in this area is well beyond a single review; we do note that many recent contributions in this area can be found on the Web.

Our work in recent years has looked at the navigation problem from two perspectives:

1. Given a user query, and no other information about the user, is it possible to semiautomate the navigation process? For a search engine the unit of information that is manipulated is a single Web page. We investigate the possibility of a trail being the logical unit of information, through the development of a navigation engine where trails are manipulated as first-class objects.
2. Given a (continuous) log of users' navigation sessions, is it possible to provide these users with navigational assistance using Web usage mining techniques? Such assistance should be personalised and have the ability to suggest users with relevant links to follow.

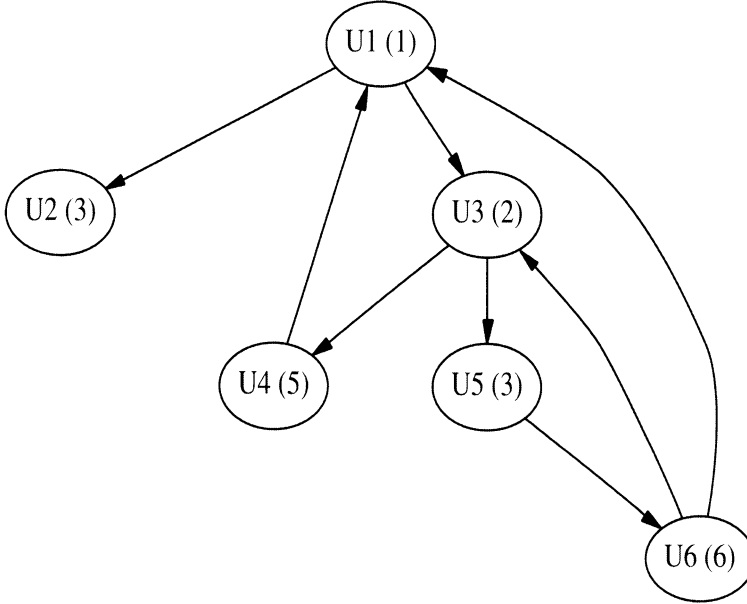
We review the following solutions we have been developing for the navigation problem:

1. The *potential gain*, which is a query independent measure of how "good" a Web page is as a starting point for navigation (see Sect. 6).
2. The *Best Trail* algorithm [70, 71], which is an algorithm for finding relevant and compact trails given a user query (see Sect. 7).
3. Data mining of user navigation patterns [8, 9, 10] based on a novel model of trail records is suitable both for virtual and physical spaces (see Sect. 8).

## 2 A Model for Web Navigation

We now introduce the main points of our model via an example. Consider the Web topology shown in Fig. 1, where each node is annotated with its URL (Uniform Resource Locator),  $U_i$ , which is the unique address of the page  $P_i$  represented by

the node. In addition to the URL  $U_i$ , each node contains the score that is a measure of the relevance of the page  $P_i$  to the input query. (We assume that a user's query represents his or her information need.) Thus, in the hypertext tradition [51], the Web is represented as a labelled directed graph [13], which we refer to as the *Web graph*.



**Fig. 1.** An example Web graph

A *trail* of information through the Web graph consists of a sequence of pages visited by the user in a navigation session. For example, with respect to Fig. 1 four possible user trails starting from  $P_1$  are:

1.  $P_1 \rightarrow P_2$ ,
2.  $P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_1 \rightarrow P_2$ ,
3.  $P_1 \rightarrow P_3 \rightarrow P_5 \rightarrow P_6 \rightarrow P_1$  and
4.  $P_1 \rightarrow P_3 \rightarrow P_5 \rightarrow P_6 \rightarrow P_3 \rightarrow P_4$ .

In our formal model we view the Web as a finite automaton called a *Hypertext Finite Automaton* (HFA), whose states are Web pages and whose transitions are links [38]. In an HFA all states can be initial and final, since we allow navigation to start and finish at any page. The state transitions of the HFA occur according to the links of the Web graph, namely the state transition from state  $s_i$  to state  $s_j$ , labelled by symbol (page)  $P_i$ , is given by

$$s_i \xrightarrow{P_i} s_j$$

and corresponds to a link from page  $P_i$  to page  $P_j$ . Our interpretation of this state transition is that a user browsing  $P_i$  decides to follow the link leading to page  $P_j$ . At

the end of the navigation session, after some further state transitions, the user will be in state, say  $s_k$ , browsing page  $P_k$ .

A word that is accepted by an HFA, which we call a *trail* of the HFA, is a sequence of pages

$$P_1 \rightarrow P_2 \rightarrow \dots \rightarrow P_n,$$

which were browsed during a navigation session, starting at page  $P_1$ , then following links according to the state transitions of the HFA and ending at page  $P_n$ . The *language* accepted by an HFA is the set of trails of the HFA. In other words, the language accepted by an HFA is the set of all possible trails a user could follow, which are consistent with the topology of the Web graph.

Let  $xy$  denote the concatenation of the words  $x$  and  $y$ . Then a word  $y$  is a *subword* of a word  $w$  if  $w = xyz$  for some words  $x$  and  $z$ , and a word  $w$  is the *join* of words  $xy$  and  $yz$  if  $w = xyz$  and  $y$  is not the empty word. In [39] we provide a characterisation of the set of languages accepted by an HFA as the subset of regular languages closed under the operations of *subwords* and *join*. This result is intuitive in terms of Web navigation since subwords correspond to *subtrails*, and the join of two words corresponds to the join of two navigation trails, where the second trail completes the first one.

We advocate the support of trails as first-class objects, and thus define a keyword-based query language, compatible with the usual input to search engines, where a *trail query* is of the form

$$k_1 k_2 \dots k_m$$

having  $m \geq 1$  keywords.

A trail  $T$  that is accepted by an HFA *satisfies* a trail query if for all  $k_i$  in the query there is a page  $P_j$  in  $T$  such that  $k_i$  is a keyword of  $P_j$ . We note that in the special case when the trail has a single Web page then all the keywords must be present in this page, complying with the usual semantics of search engine query answering. (We discuss trail scoring mechanisms in Sect. 7.) In [38] we show that checking whether an HFA accepts a trail satisfying a trail query is NP-complete. The proof of this result utilises a duality between *propositional linear temporal logic* and a subclass of finite automata. In temporal logic terminology the condition that  $k_i$  is a keyword of page  $P_j$  is the assertion that “sometimes”  $k_i$ , viewed as a condition on  $P_j$ , is true. Therein we also defined a more general class of trail queries which supports the additional temporal operators “nexttime” and “finaltime”, and more general Boolean conditions. In the context of the Web the natural interpretation of “time” is “position” within a given trail. So, “sometimes” refers to a page at some position in the trail, “nexttime” refers to the page at the next position in the trail, and “finaltime” refers to the page at the last position in the trail. In [38] we showed that only for restricted subclasses of trail queries is the problem of checking whether an HFA accepts a trail satisfying a trail query polynomial-time solvable. Such a subclass essentially prescribes a one-step-at-a-time navigation session using the “nexttime” operator. Current navigation practice where links are followed one at a time conforms to this subclass.

These time-complexity results have led us to enrich the semantics of HFA by attaching probabilities (or equivalently weights) to state transitions resulting in *Hypertext Probabilistic Automata* (HPA) [39]. The transition probabilities in our model can have two interpretations. First, they can denote the proportion of times a user (or a group of users) followed a link, and second, they can denote the relevance (or expected utility) of following a link. The first interpretation is developed in Sect. 8 while the second is developed in Sect. 7.

We further develop the notion of HPA by viewing them as finite *ergodic Markov chains* [30]. In order to realise this view we may consider the user's home page as an artificial starting point for all navigation sessions and assume that there is a positive probability (however small) of jumping to any other relevant Web page. We can thus modify Fig. 1 by adding to the graph the user's home page and links from it to all other pages. The probabilities of these links are the initial probabilities of the Markov chain. Moreover, we assume that the user is following links according to the transition probabilities and when completing a navigation session returns to his or her home page. Thus we would further modify Fig. 1 by adding links from existing pages to the artificial home page. A probability attached to such a link denotes the probability of concluding the navigation session after visiting a particular Web page. The resulting HPA can be seen to be an ergodic Markov chain [40]. The probability of a trail  $T$  is thus defined as the product of the initial probability of the first page of the trail together with the transition probabilities of the links in the trail.

As a further example, consider the Web graph shown in Fig. 2, which shows a fragment of the School of Computer Science and Information Systems (SCSIS) Web site; see [www.dcs.bbk.ac.uk](http://www.dcs.bbk.ac.uk). Note that the logical organisation of the Web site gives rise to many meaningful trails such as

$$\text{SCSIS} \rightarrow \text{Research} \rightarrow \text{Students} \rightarrow \text{Kevin},$$

which is intuitive and easy to understand.

In our model we can distinguish between the following different types of trails according to their mode of creation:

1. *authored trails*: trails that have been explicitly defined for a given purpose; for example a guided tour [68, 63] through the research areas of a computer science department.
2. *derived trails*: trails that are derived according to specific criteria; for example, as a result of query (see Sect. 7) or following the hierarchy (or more generally, a suitable ontology) of a Web directory such as Yahoo or the Open Directory.
3. *emergent trails*: trails that are created via repeated navigation and browsing through a Web space (see Sect. 8). These may be
  - a) *personal trails*: trails that arise from an individual's activity, or
  - b) *collaborative trails*: trails that arise from a group activity through the space.

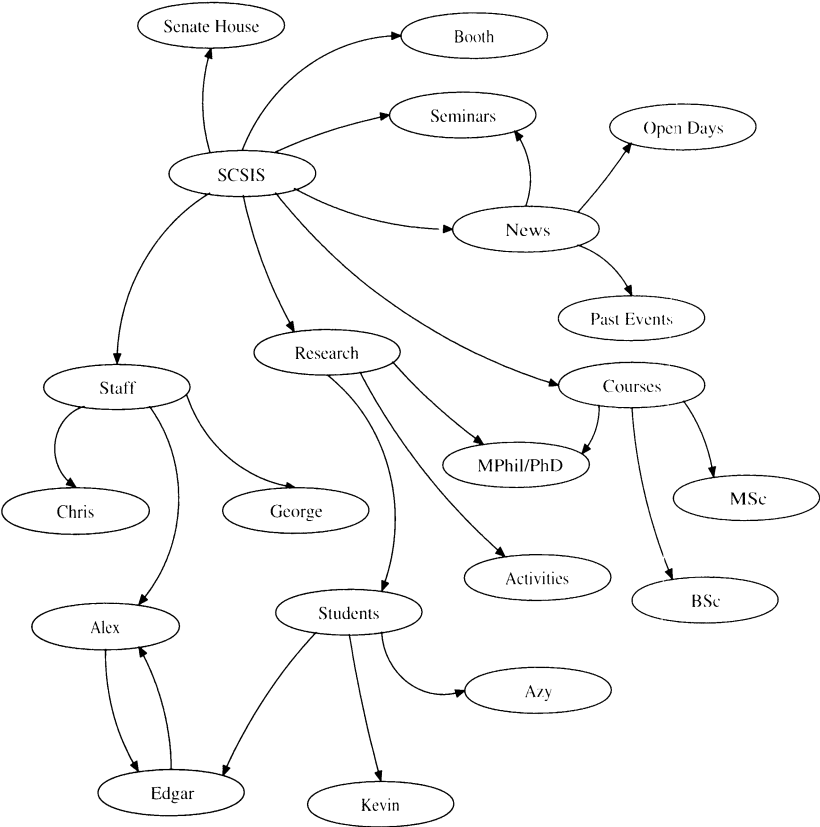


Fig. 2. SCSIS Web graph

### 3 An Historical Perspective of Navigation in Hypertext

The inspiration for hypertext comes from the *memex* machine proposed by Bush [14] (see [53] for a collection of essays on Bush and his memex). The memex is a “sort of mechanized private file and library” which supports “associative indexing” and allows navigation whereby “any item may be caused at will to select immediately and automatically another”. Bush emphasises that “the process of tying two items together is an important thing”. By repeating this process of creating links we can form a *trail* which can be traversed by the user, in Bush’s words “when numerous items have been thus joined together to form a trail they can be reviewed in turn”. The motivation for the memex’s support of trails as first-class objects was that the human mind “operates by association” and “in accordance to some intricate Web of trails carried out by the cells of the brain”.

Bush also envisaged the “new profession of trailblazers” who create new trails for other memex users, thus enabling sharing and exchange of knowledge. The memex was designed as a personal desktop machine, where information is stored locally on the

machine. Trigg [68] emphasises that Bush viewed the activities of creating a new trail and following a trail as being connected. Trails can be authored by trailblazers based on their experience (these are authored trails) and can also be created by memex which records all user navigation sessions (these are emergent trails). In his later writings on the memex, published in [53], Bush revisited and extended the memex concept. In particular, he envisaged that memex could “learn from its own experience” and “refine its trails”. By this Bush means that memex collects statistics on the trails that the user follows and “notifies” the ones which are most frequently followed. Oren [54] calls this extended version *adaptive memex*, stressing that adaptation means that trails can be constructed dynamically and given semantic justification; for example, by giving these new trails meaningful names.

Engelbart’s On-Line System (NLS) [23] was the first working hypertext system, where documents could be linked to other documents and thus groups of people could work collaboratively. The term “hypertext” was coined by Ted Nelson in 1965 (see [50]), who considered “a literature” (such as scientific literature) to be a *system of interconnected writings*. The process of referring to other connected writings when reading an article or a document is that of *following links*. Nelson’s vision is that of creating a repository of all the documents that have ever been written thus achieving a universal hypertext. Nelson viewed his hypertext system, which he called *Xanadu*, as a network of distributed documents that should be allowed to grow without any size limit and such that users, each corresponding to a node in the network, may link their documents to any other documents in the network. Xanadu can be viewed as a generalised memex system, which is both for private and public use. As with memex, Xanadu remained a vision which was not fully implemented. Nelson’s pioneering work in hypertext is materialised to a large degree in the Web, since he also viewed his system as a means of publishing material by making it universally available to a wide network of interconnected users. An interesting feature of Xanadu is its copyright mechanism, where reuse of material is done through linking to the portion of material being republished. Berners-Lee turned the vision of hypertext into reality by creating the World Wide Web as we know it today [6] through the invention of the URL, HTTP and HTML, and more recently through the semantic Web and XML.

## 4 Tackling the Navigation Problem

We have already introduced the navigation problem in Sect. 1 whereby users “get lost in hyperspace” while they are surfing, as a result of losing the context in which they are browsing, and are then unsure how to proceed in terms of satisfying their information need. We will now briefly survey some recent suggestions for tackling this problem; we will defer discussion of our proposal via the Best Trail algorithm to Sect. 7.

Search engine results are not always up-to-date, since they only access information stored in a static index. The idea of *dynamic* search is to fetch Web pages online during the search process thus guaranteeing valid and precise information. The downside of

such a dynamic approach is that it does not scale. An early dynamic search algorithm called *fish search* [20] uses the metaphor of a school of fish (search agents) foraging (searching) for food (relevant documents). When food is found, the fish reproduce and continue looking for food. In the absence of food or when the water is polluted (poor bandwidth), they die. An improved algorithm is *shark search* [26], which uses the vector-space model [59] to detect relevant documents, and a decay factor between zero and one to reduce the influence of pages which are further away from the starting point. The decay factor can also be viewed as taking into account the cost to the user of following an additional link [44]. A further improvement in the shark search algorithm is to give priority to anchor text attached to links and its broader textual context, in determining the relevance of documents that may be followed by clicking on a link.

The spread of activation algorithm [56] simulates users' surfing patterns when they are foraging for relevant information at some Web locality, i.e. a collection of related Web pages, in an attempt to understand how surfers allocate their time in pursuit of information. The activation network is represented as a graph whose nodes are Web pages and where each link has a strength of association attached to it. The strength of association of a link may indicate textual similarity between pages or, alternatively, usage statistics, i.e. the number of users following the link. The activation level of pages is represented by a vector which evolves over time and decays at each time step to reduce the influence of pages according to their distance from an initial set of activated pages. In [56] the algorithm is used to find textually similar pages within a locality and the most frequently browsed pages in a Web site of home page visitors.

Related to the above work is that of information foraging by navigation, i.e. surfing along links, and picking up proximal cues, such as snippets of text, to assess distant content which is only revealed after following one or more links. The *scent of information* is the perception of the value and cost obtained from these proximal cues representing distant content. In [16] various techniques have been developed to predict information scent based on usage mining analysis, content and link topology. In particular, a technique called *Web user flow by information scent* has been developed, which simulates agents navigating through a Web site, to better understand how users navigate and to predict when their information need is met. In this technique the agents have information needs described by a simple textual description such as "research groups in the computer science department" and, as in the foraging model, the scent at a given page is evaluated by comparing the user's need with the information scent associated with linked pages. The navigation decisions based on the information scent are stochastic, so more agents follow higher scent links. Agents stop surfing when they either find a target page or they have expended a given amount of effort.

Web-Watcher [29] is an automated tour guide for the Web. Web-Watcher accompanies users as they are browsing and suggests to them relevant links to follow based on its knowledge of users' information needs. At the beginning of the tour each user types in some keywords to describe their information need. Whenever a user follows a link

the description of this link, which is initially just its anchor text, is augmented by the user's keywords. Thus links accumulate keywords and Web-Watcher can recommend to a user the link description that best matches his or her keywords describing the information need, where similarity is measured using the vector-space model. A complimentary learning method used by Web-Watcher is based on reinforcement learning with the objective of finding trails through the Web site that maximise the amount of relevant information encountered when traversing the path. More specifically, the technique is based on Q-learning [65], which chooses a state such that the discounted sum of future rewards is maximised. In this application the states are Web pages and the reward is the score returned for the Web page the user is browsing, with respect to keywords the user initially specified; again the score is computed using the vector-space model.

An adaptive agents approach to satisfying a user's information need, called InfoSpiders search, was proposed in [47]. InfoSpiders search is an online dynamic approach, as is the shark search algorithm, the motivation being that traditional search is limited by static indexes that are incomplete and often out of date. In this approach a population of agents navigates across Web pages and makes autonomous decisions about which links to follow next. Initially an information need is specified as a set of keywords along with a set of starting pages. Each agent is then positioned at one of the starting points and given an amount of energy that can be consumed. Each agent situated at a page makes a decision which link to follow based on the current document content, which is used to estimate the relevance of neighbouring pages that can be reached by link traversal. An agent performs its decision using a local feed-forward neural network, which has an input node for each initial keyword and one output node for the relevance. When an agent moves to the next page by following a link its energy level is updated and a reward, in the form of additional energy, is given to the agent if the page reached turns out to be relevant; the agent incurs a cost each time it accesses a document thereby reducing its energy level. As in Web-Watcher, InfoSpiders adapts its behaviour using Q-learning. After each reinforcement step an agent either replicates or dies according to its energy level. Menczer and Belew [47] conducted several experiments to test InfoSpiders and concluded that their algorithm performed better than traditional search algorithms such as breadth-first search and best-first-search. InfoSpiders is seen to add value to search engines: the search engine can provide "good" starting points and InfoSpiders can reach fresh pages in the neighbourhood of the starting points, which may not have been indexed by the search engine.

In [24] issues of navigation in Web topologies are explored in terms of a *viewing graph*, which is a small subgraph of the hypertext structure in which the user is currently navigating. Navigability is the property of being able to find the shortest path to a target node from the node currently being browsed by making decisions based solely on local information visible at the current node. This implies that at each node in the viewing graph sufficient information must be available to guide the user to the correct target node via the shortest route. Moreover, the information available at each node must be compact. Under this definition of navigability, navigation on the Web is, in



general, not effective, due to the fact that local information at nodes is limited. Ways of improving navigation on the Web include organisation of information into classification hierarchies and the ability to make local decisions through similarity-based measures between nodes of close proximity. Examples of classification hierarchies are Yahoo and the Open Directory, and an example of a similarity-based measure, mentioned above, is the similarity of link text to a user query.

## 5 Navigation Tools

Here we concentrate on navigation aids that help surfers orient themselves within the graph topology and find their way. The *browser* is the component of a hypertext system that helps users search for and inspect the information they are interested in by graphically displaying the relevant parts of the topology and by providing contextual and spatial cues with the use of *orientation tools*. We have taken a wider view of the browser than currently implemented Web browsers. In an interview (*Wired News*, 14 February 2003) Marc Andreessen, one of the founders of Netscape, said:

“If I had to do it over again, I’d probably show some sort of graphical representation of a tree, so you could see what path you’re travelling on and could backtrack. I’d also include thumbnail renderings on the tree to show where you’d been.”

A simple orientation tool is the *link marker*, which acts as a signpost to tell the user what links can be immediately followed and what links have recently been traversed. In the context of HTML, link text is highlighted and should give accurate information about the page at the other side of the link; so link text such as *click here* is meaningless as it does not convey any information to the user. Another useful orientation tool is the *bookmark*, allowing readers to mark a page to which they can return to on demand when feeling lost [7]. All Web browsers provide a bookmark facility, which allows users to view the titles of the Web pages on the list, normally through a pull-down menu, and load any one of these pages into the browser. Although bookmarks are useful, it has been found that the rate of page addition is much higher than the rate of page deletion, implying that users have problems in managing their bookmarks [17]. Readers may, in principle, also mark pages which were already visited in order to avoid repetition; such marks are called *bread crumbs* [7]. In the context of Web browsers there is a primitive notion of bread crumbs when the colour of a link that has been clicked on is changed.

*Maps* (or *overview diagrams*) give readers a more global context by displaying to them links which are at a further distance than just one link from the current position. For instance, current Web sites often provide a *site map* to give visitors an overview of the contents of the site. Maps can be displayed using a *fish-eye view* that selects information according to its *degree of interest*, which decreases as the page under consideration is further away from the currently browsed page [67]. An ambitious kind of site map, based on the fish-eye concept, is provided by the hyperbolic

browser [35], which allows the user to dynamically focus on different parts of the Web site by using a novel visualisation technique based on hyperbolic geometry; see [www.inxight.com](http://www.inxight.com).

A set of tools that aid the construction of maps by performing a structural analysis of the graph topology is described in [57]. One such tool is an hierarchical structure that can be imposed on the Web graph, where its root is chosen to be a central node whose distance to other nodes is relatively small. Another tool creates semantic clusters within the Web graph by identifying strongly connected components of the graph [13]. *Landmark nodes* are prominent nodes within a Web site, or more generally, nodes within the subspace the user is browsing through. A simple formula for discovering landmark nodes in the Web graph based on the number of pages that can be reached from a page or that can reach the page when following at most two links was proposed by [48]. Once the landmark nodes are known, the context of the page that the user is currently browsing can be shown by its relationship to nearby landmark nodes. A survey covering additional metrics based on Web page content and link analysis can be found in [21].

In [32] the activity of user navigation within a hypertext is compared to the activity of *wayfinding* through a physical space, where wayfinding is defined as the process used to orient and navigate oneself within a space, the overall goal of wayfinding being to transport oneself from one place to another within the space. Both activities, in a virtual space and a physical one, include user tasks such as being aware of one's current location, planning a route to follow and executing the plan. Research into wayfinding in physical spaces is based upon the assumption of the existence of cognitive maps encoding the user's knowledge about the space he or she is navigating through. Such spatial knowledge can be classified into the representations of place, route and survey knowledge, which concerns the spatial layout of the salient places. Various hypertext tools that aim to help solve the disorientation problem have been developed which are inspired by the spatial metaphor. These include differentiation of regions, maps, guided tours, landmark nodes, fisheye views, history lists, history trees and summary boxes.

An orientation tool that has been developed within the hypertext community is the *guided tour*, which actively guides users through the space being navigated by suggesting interesting trails that users can follow [68]. One such system, called Walden's paths [63], allows teachers to create annotated trails of Web pages for their students to follow and browse. There remains the question of whether trail creation can be automated, at least partially, as is further investigated in Sect. 7. A dynamically created trail that highlights the path the user has followed within a Web site is the *navigation bar* advocated by Nielsen [52]. For example,

SCSIS → Research → Activities → Database and Web Technologies Group

would indicate to the visitor the trail he or she has followed within the School's Web site from the home page to the Database and Web Technologies group (see Fig. 2). By highlighting the navigation history within the Web site the visitor can easily *backtrack* his or her steps to a previously browsed page. This idea can be refined by using a

side-bar to highlight links which provide navigation options to the visitor from the current page they are browsing. One can take this a step further and provide dynamic personalised navigation cues; see [43].

Two standard navigation tools provided by Web browsers are the *back button* and the *history list*. Another simple navigation aid is the *home* button, which allows users to jump to their home page at any time. The back button is a stack-based mechanism allowing the user to retrace their trail one page at a time, and the complimentary forward button returns to the page browsed before the last invocation of the back button. The history list contains the sequence of Web pages that were browsed by the user and can be accessed sequentially according to the time browsed or some other criteria such as the most visited page. Current browsers also provide a search facility over the history list. The history list is displayed linearly, although in practice Web pages act as branching points, for example, users often start navigating from a home page of a site and take different routes according to their information need.

Two studies carried out in 1996 [66] and in 1999 [17] investigated how Web surfers use the navigation tools provided by Netscape Navigator. Tauscher and Greenberg [66] found that, by far, the most used navigation tool was the back button. Other mechanisms such as the forward button and history list were used infrequently as a percentage of the total number of user actions. They calculated the recurrence rate as

$$\frac{\text{total number of URLs visited} - \text{different number of URLs visited}}{\text{total number of URLs visited}},$$

and discovered that this rate was close to 60%. It was also found that there is about a 40% chance that the next page visited is within the last six pages visited. Overall the back button is the dominant mechanism used to revisit pages. It is simple and intuitive to use but inefficient in retrieving distant pages. The authors also found that the history and bookmark mechanisms are used much less than the back button. In a follow-up study Cockburn and McKenzie [17] found the recurrence rate to be much higher at about 80% and that almost all users had one or two pages that they revisited far more often than others; this could be, for example, their home page or their favourite search engine.

The stack-based behaviour of the back button is incompatible with the navigation history of the user as the following example shows. Suppose the user navigates as follows (see Fig. 2):

SCSIS  $\rightarrow$  Staff  $\rightarrow$  *Back to SCSIS*  $\rightarrow$  Research  $\rightarrow$  *Back to SCSIS*,

where *Back to SCSIS* indicates the use of the back button. The stack-based semantics means that Staff is *inaccessible* to the user through the back button, since when the user clicked on the back button the first time all pages above it in the stack were removed. Despite the semantics of the back button being misunderstood by many users it is still heavily used. Cockburn et al. [18] conducted a further study evaluating a history-based behaviour of the back button as opposed to the standard stack-based

behaviour; so in the previous example, with history-based behaviour, a further click on the back button would bring the user to the Staff page. They concluded that from the user's point of view there was no significant difference between the stack and history behaviours. Moreover, the history-based behaviour is inefficient in returning to parent pages from a deeply nested page, but for distant navigation tasks it was highly efficient.

An interesting navigation tool is the toolbar recently introduced by Google, which can be installed within Microsoft's Internet Explorer; see <http://toolbar.google.com>. It provides the user with various options, allowing the user to access the search engine directly from the browser, either searching the Web as a whole or just searching within the Web site the user is currently at. As long as privacy and security issues are dealt with, we believe in the potential of adding navigation tools to the browser.

Web directories such as Yahoo and the Open Directory organise information according to a categorisation, so, for instance, *Web usability* can be found by navigating the following path in the Open Directory,

Computers → Internet → Web Design and Development → Web Usability,

where we find several subcategories such as *accessibility*, several related categories such as *human-computer interaction* and many Web pages that were manually categorised. (In principle, it is possible to automate, or at least semiautomate, the categorisation process but this problem is not within the scope of this chapter; see [61].) Hearst [25] argues that search engines should incorporate category information into their search results to aid navigation within Web sites. Moreover, to help cut down the number of possible trails a visitor can follow, the user interface as well as the site's structure should reflect the tasks that the visitor is attempting to accomplish. Hearst points to the potential use of category metadata to help achieve a better integration between the user's information need and the site's structure by dynamically determining the appropriate categories for a given query.

## 6 The Navigation Potential of a Web Page

Although, as far as we know, Web search engines weight home pages higher than other pages, they do *not* have a general mechanism to take into consideration the navigation potential of Web pages. Our aim in this section is to propose a metric for finding "good" starting points for Web navigation, which is independent of the user query. Once we have available such a metric we can weight this information into the user query so that the starting points will be both relevant and have navigational utility. From now on we will refer to the navigability measure of a Web page as its *potential gain*. We note that the application that initially led us to investigate the potential gain was to provide starting points for the search and navigation algorithm described in the next section, but we believe that this notion has wider applicability within the general context of navigation tools.

In the following let the  $G$  represent the Web graph,  $G$  having a set of pages (or URLs identifying the pages)  $N$  and a set of links  $E$ . Starting URLs should satisfy the following criteria:

1. They should be *relevant*, i.e. their score with respect to the user's goal should be high.
2. They should be *central* [13] in the sense that their distance to other pages is minimal.
3. They should be *connected* in the sense that they are able to reach a maximum number other pages. (If  $G$  is strongly connected, then this clause is redundant.)

Now, let  $links(U_i)$  be a function that returns the collection of links out-going from  $U_i$ . Algorithm 1, computes the potential gain of all the nodes in the node set,  $N$ , of a Web graph  $G$ . It has two additional parameters: (i)  $MAX$ , defining the maximal depth in the breadth-first traversal of the Web graph; and (ii)  $\delta(d)$ , which is a monotonically decreasing function of the depth  $d$ . Two reasonable such functions are the reciprocal function  $1/d$  and the discounting function  $\gamma^{d-1}$ , where  $0 < \gamma < 1$ . The justification for these functions is that the utility of a page diminishes with the distance of the page from the starting point. This assumption is consistent with experiments carried out on Web data sets [28, 37].

The algorithm also involves a constant  $C$  between 0 and 1, which is the lower bound potential gain of any pages in  $N$ ; we will conveniently take  $C$  to be 1. The algorithm outputs an array PG where  $PG[U_i]$  is the potential gain of the URL  $U_i$  in  $N$  computed to depth  $MAX \geq 1$ .

**Algorithm 1 (Potential\_Gain( $G, MAX, \delta$ ))**

```

1. begin
2.   for each URL  $U_i$  in  $N$ 
3.      $PG[U_i] \leftarrow C$ ;
4.      $prev\_count[U_i] \leftarrow 1$ ;
5.   end for
6.   for  $d = 1$  to  $MAX$ 
7.     for each URL  $U_i$  in  $N$ 
8.        $cur\_count[U_i] = \sum_{U_k \text{ in } links(U_i)} prev\_count[U_k]$ ;
9.        $PG[U_i] \leftarrow PG[U_i] + \delta(d) \cdot cur\_count[U_i]$ ;
10.    end for
11.    for each URL  $U_i$  in  $N$ 
12.       $prev\_count[U_i] \leftarrow cur\_count[U_i]$ ;
13.    end for
14.  end for
15.  return PG;
16. end.

```

We note that in practice we will need to normalise the potential gain values as, in principle, they may increase without bound. For the discounting function we can guarantee convergence as  $MAX$  tends to infinity if the product of  $\gamma$  and the maximum out-degree of a Web page is less than one. On the other hand, for the reciprocal function no such convergence is guaranteed. The potential gain of a starting Web page can be seen to depend on the number of trails out-going from the Web page, where the value of the trails diminish with distance. We further note that Algorithm 1 can be described much more concisely using matrix–vector notation, as:

$$\begin{aligned} \text{cur\_count} &= G \cdot \text{cur\_count}, \\ \text{PG} &= \text{PG} + (\delta(d) \cdot \text{cur\_count}), \end{aligned}$$

where the above equations are iterated  $MAX$  times.

A complementary metric to the potential gain is the *gain rank*, which measures the likelihood of navigating to a given Web page. Its computation can be obtained by replacing in Algorithm 1  $\text{links}(U_i)$  by  $\text{inlinks}(U_i)$ , where  $\text{inlinks}(U_i)$  is a function that returns the collection of links going into  $U_i$ . It would be interesting to compare the gain rank with Google’s PageRank metric [55].

In Table 1 we show the unnormalised potential gain and gain rank values computed to depth  $MAX = 100$  of the example Web graph shown in Fig. 2, using the reciprocal function in the calculation. It can be seen that the home page of the Web graph, i.e. SCSIS, has the highest potential gain within the Web graph, followed by the pages: Research, Students and Staff. On the other hand, the gain rank of the pages Alex and Edgar have the highest gain rank within the Web graph, indicating that they are reachable through more trails than other pages.

## 7 The Best Trail Algorithm

We present an algorithm for deriving relevant trails from the Web graph given a user query. The algorithm, called the *Best Trail*, semiautomates navigation by probabilistically constructing a tree whose most relevant trails are presented to the user. The algorithm is adaptive in the sense that it dynamically searches for the preferred trails by mimicking a user navigation session and scoring the trails as they are expanded according to the topology of the Web site.

Prior to presenting the algorithm we discuss the issue of scoring the relevance of trails as first-class objects. As before, we let the  $G$  represent the Web graph,  $G$  having a set of pages (or URLs identifying the pages)  $N$  and a set of links  $E$ . Every link connects two pages: its starting page is called the *anchor*, and its finishing page is called the *destination*.

We interpret the score  $\mu(m)$  of a Web page  $m$  in  $N$  as a measure (or utility) of how relevant  $m$  is to the user. In this sense we cater for personalisation, and we would expect  $\mu$  to be different for distinct users. Another interpretation of  $\mu$  is that

URL	Title	Out-degree	In-degree	Potential gain	Gain rank
1	SCSIS	7	0	22.0414	1.0000
2	Booth	0	1	1.0000	2.0000
3	Senate house	0	1	1.0000	2.0000
4	Seminars	0	2	1.0000	3.5000
5	News	3	1	4.0000	2.0000
6	Open days	0	1	1.0000	2.5000
7	Past events	0	1	1.0000	2.5000
8	Courses	3	1	4.0000	2.0000
9	MSc	0	1	1.0000	2.5000
10	BSc	0	1	1.0000	2.5000
11	MPhil/PhD	0	2	1.0000	4.0000
12	Research	3	1	9.1874	2.0000
13	Activities	0	1	1.0000	2.5000
14	Students	3	1	8.1874	2.5000
15	Azy	0	1	1.0000	2.8333
16	Kevin	0	1	1.0000	2.8333
17	Edgar	1	2	6.1874	17.4999
18	Staff	3	1	8.1874	2.0000
19	Chris	0	1	1.0000	2.5000
20	George	0	1	1.0000	2.5000
21	Alex	1	2	6.1874	17.3117

**Table 1.** Potential gain and gain rank values for example

it is query specific and returns the relevance of a page with respect to a given query, where the query is viewed as representing the goal of the navigation session. In this interpretation  $\mu$  can be viewed as the scoring function of a search engine with respect to a given query. In both interpretations of  $\mu$  the user initiating a navigation session would like to maximise the relevance (or suitability) of the trail to the query. The relevance of a trail

$$T = U_1 \rightarrow U_2 \rightarrow \dots \rightarrow U_n$$

is realised by its *score*, which is a function of the scores of the individual Web pages of the trail; we denote the scoring function of a trail by  $\rho$ . Five reasonable scoring functions for a trail are:

1. The average score of the URLs in the trail, i.e.

$$\rho(T) = \text{avg}(T) = \frac{\mu(U_1) + \mu(U_2) + \dots + \mu(U_n)}{n}.$$

2. The average score of the distinct URLs in the trail, i.e. for the purpose of obtaining the scoring of the trail, each URL in the trail is counted only once. In this case  $\rho$  is denoted by *avg\_dist*.
3. The sum of the scores of the distinct URLs in the trail divided by the number of pages in the trail; this scoring function penalises the trail when a URL is visited more than once. In this case  $\rho$  is denoted by *sum\_dist*.

4. The sum of the discounted scores of the URLs in the trail, where the discounted score of  $U_i$ , the URL in the  $i$ th position in the trail, is the score of  $U_i$  with respect to the query multiplied by  $\gamma^{i-1}$ , where  $0 < \gamma < 1$  is the discount factor. The discounted score of  $T$  is given by

$$\rho(T) = \text{discount}(T) = \sum_{i=1}^n \mu(U_i) \gamma^{i-1}.$$

5. The maximal score of all of the URLs in the trail. In this case  $\rho$  is denoted by  $\max$ .

We can also combine scoring functions 3 and 4 by discounting in function 3 each URL according to its previous number of occurrences within the trail. (This combination of scoring functions, in addition to *sum\_dist*, are the ones we have used in the navigation system we are developing, which is discussed towards the end of this section.) We observe that all the trail scoring functions we have defined are bounded due to the definition of  $\mu$  and the fact that  $N$  is finite, and as a result an important property of the above trail scoring functions is that they define convergent sequences.

We now describe the algorithm assuming one URL as its starting point although, in general, the algorithm will take as input several starting points and compute the best trail for each one of them; see the pseudocode of Algorithm 2. Starting from the initial URL, the algorithm follows links from anchor to destination according to the topology of the Web, that is, when an out-link exists from the Web page identified by its URL then it may be traversed by the algorithm.

The algorithm builds a *navigation tree* whose root node is labelled by the URL of the starting point. Each time a destination URL is chosen a new node is added to the navigation tree and is labelled by the destination URL. Nodes that may be added to the navigation tree as a result of traversing a link that has not yet been followed from an existing node are called *tip nodes* (or simply tips). We also consider the special case when a link has been traversed to a destination URL and the page associated with this URL has no out-links. Nodes in the navigation tree which are labelled by such URLs are called *sinks* and are also considered to be tip nodes.

At any given stage of the running of the algorithm each tip node in the current state of the navigation tree is considered to be a destination of an anchor of a link to be followed; in the case when the tip node is a sink we can consider the destination to be the sink itself. The algorithm uses a random device to choose a tip node to be added to the navigation tree; in the special case when the tip node is a sink the navigation tree remains unchanged. The weight that is attached to a tip node for the purpose of the probabilistic choice is proportional to the score of the trail induced by the tip node, which is the unique sequence of URLs labelling the nodes in the navigation tree forming a path from the root node of the tree to the tip node under consideration. (The exact formula for calculating the probability of a tip node is given below.) We call the process of adding a tip node to the navigation tree *node extension*. The Best Trail algorithm terminates after a prescribed number of node extensions, each such extension being a single iteration within the algorithm.



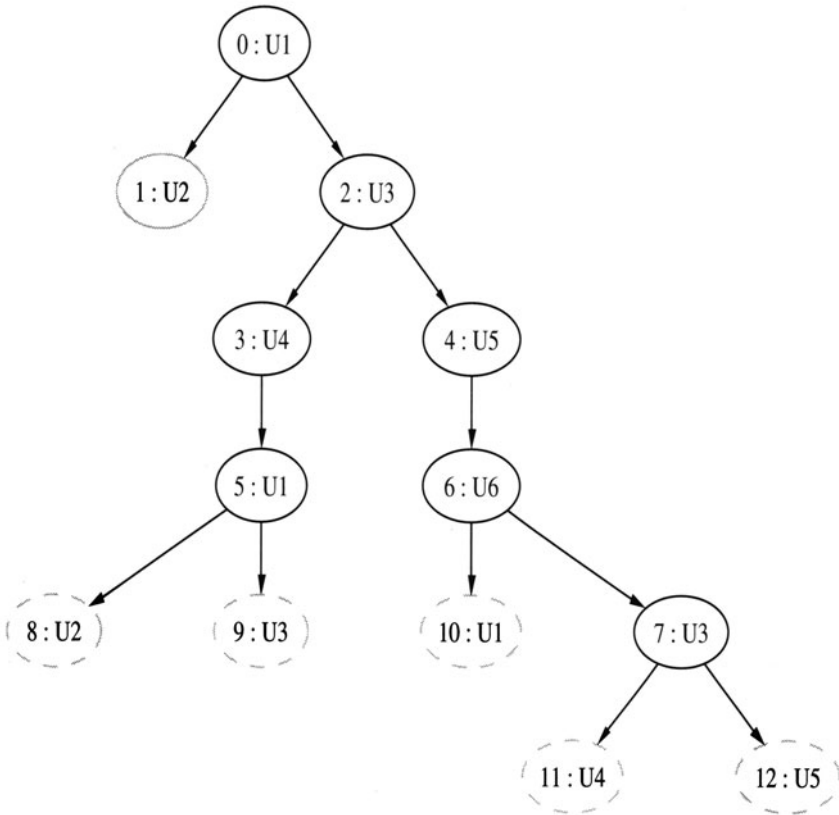
The algorithm has two separate stages, the first being the *exploration stage* and the second being the *convergence stage*. Each stage comprises a preset number of iterations. During the exploration stage a tip node is chosen with probability purely proportional to the score of the trail that it induces. During the convergence stage we apply a “cooling schedule”, where tip nodes which induce trails having higher scores are given exponentially higher weights at each iteration according to the rank of their trails, as determined by their trail scores, and the number of iterations completed so far in the convergence stage. A parameter called the *discrimination factor*, which is a real number strictly between zero and one, determines the convergence rate. When the algorithm terminates the *best trail* is returned, which is the highest ranking trail induced by the tip nodes of the navigation tree. The convergence of the algorithm to the absolute best trail is guaranteed, provided the number of iterations in both stages of the algorithm is large enough and the discrimination factor is not too low. The Best Trail algorithm can be modified so that the discrimination factor decreases dynamically during the convergence stage.

We now define the terminology used for the Best Trail algorithm, given a starting URL, say  $U$ , of the current navigation session.

1. The *unfolding* of  $G$  is a possibly infinite tree having root  $U$ , resulting from traversing  $G$  starting from  $U$  in such a way that each time a URL is revisited during the traversal of  $G$  a duplicate of this URL is added to the node set of the unfolding of  $G$ . Thus duplicates of a URL resulting from multiple visits result in distinct nodes in the unfolding of  $G$  although their individual scores are identical. (Note that the unfolding of  $G$  is finite if and only if  $G$  is acyclic.)
2. A *navigation tree* having root  $U$  is a finite subtree of the unfolding of  $G$ .
3. A *frontier* node of a navigation tree is either
  - a) a leaf node in the tree, or
  - b) a node, say  $m$ , in the tree associated, say with URL  $U_i$ , such that the set of URLs associated with the successors of  $m$  in the navigation tree is a proper subset of  $links(U_i)$ , i.e. there is a link in  $E$  with anchor  $U_i$  that has not yet been traversed during the current navigation session.
4. A *tip* node in a navigation tree is either
  - a) a node which is associated with a sink in  $G$ , i.e. a node whose associated URL has no successors in  $G$ ; we call such a tip node a *sink* node (we note that in this case the tip node is also a frontier node which is a leaf), or
  - b) a node, say  $m$ , associated with a successor in  $G$  of one of the URLs associated with a frontier node, say  $m'$ , in the navigation tree, such that  $m$  is the destination of a link that has not yet been traversed from the URL associated with  $m'$ .
5. The *score* of a trail induced by a tip node, say  $t$ , is the score of the trail which is the unique sequence of URLs labelling the nodes in the navigation tree forming a path from the root node of the tree to  $t$ ; we overload  $\rho$  and denote this score by  $\rho(t)$ .
6. The *extension* of a navigation tree with a one of its tip nodes, which we call *node extension*, is done according to the following two cases:

- a) if the tip node is a sink node then the navigation tree remains unchanged, otherwise
- b) add a new node and edge to the navigation tree such that the anchor node of this edge is the parent frontier node of this tip node and the destination node is the tip node itself. The new node becomes a frontier node of the extended navigation tree.

Assuming that  $U_1$  is the starting URL of the Best Trail algorithm for the Web topology shown in Fig. 1, a possible navigation tree after seven node extensions is given in Fig. 3. Each node in the navigation tree is annotated with a unique number and with its URL; the tip nodes of the navigation tree have shaded boundaries. The root of the navigation tree is node 0, which is labelled by the starting URL  $U_1$ , and the nodes that were added to the navigation tree as a result of the seven node extensions are numbered from 1 to 7.



**Fig. 3.** An example navigation tree

The frontier nodes of the navigation tree are 1, 5, 6 and 7. Node 1 is also a tip node of the navigation tree since it is a sink. Node 5 is the parent of two tip nodes, numbered 8 and 9. Node 6 is the parent of a single tip node, numbered 10. Similarly, node 7 is the parent of two tip nodes, numbered 11 and 12. Table 2 shows the tips, their induced trails and the score of these trails according to the five trail scoring functions we defined in Sect. 2. As can be seen in this example, the trail to tip 11 is the highest scoring trail, irrespective of the scoring function used.

Tip	Induced trail	<i>avg</i>	<i>avg_dist</i>	<i>sum_dist</i>	<i>discount</i> ( $\gamma = 0.75$ )	<i>max</i>
1	$U_1, U_2$	2.00	2.00	2.00	3.25	3.00
8	$U_1, U_3, U_4, U_1, U_2$	2.40	2.75	2.20	6.68	5.00
9	$U_1, U_3, U_4, U_1, U_3$	2.20	2.66	1.60	6.37	5.00
10	$U_1, U_3, U_5, U_6, U_1$	2.60	3.00	2.40	7.03	6.00
11	$U_1, U_3, U_5, U_6, U_3, U_4$	3.17	3.40	2.83	8.53	6.00
12	$U_1, U_3, U_5, U_6, U_3, U_5$	2.83	3.00	2.00	8.06	6.00

**Table 2.** The trails induced by the tips and their scores

We now define several auxiliary functions and parameters used in the Best Trail algorithm, given a navigation tree  $D_i$  and a tip node  $t$  of  $D_i$ .

1. The *discrimination factor*, denoted by  $df$ , is a parameter such that  $0 < df < 1$ . Intuitively,  $df$  allows us to discriminate between “good” trails and “bad” trails by reducing the influence of trails which perform badly. Thus during the convergence stage “better” trails get assigned exponentially higher probability, taking into account the fact that the longer the history the more influence “better” trails have. This weighting guarantees that, asymptotically, the ‘best’ trail will eventually have probability one.
2.  $I_{\text{explore}} \geq 0$  is the number of iterations during the exploration stage of the algorithm.
3.  $I_{\text{converge}} \geq 1$  is the number of iterations during the convergence stage of the algorithm.
4. The *rank* of the trail induced by a tip node  $t_k$  of  $D_i$ , denoted by  $\tau(t_k)$ , is the position of  $\rho(t_k)$  within the set of scores of the trails induced by the tip nodes  $t_1, t_2, \dots, t_n$ , when the scores are arranged in descending order of magnitude and duplicate scores are removed.
5. The *probability of a tip node*,  $t$  of  $D_i$ , where  $\alpha$  is either 1 or  $df$  and  $j$  denotes either an exploration or convergence step, is denoted by  $P(D_i, t, \alpha, j)$ , and given by

$$P(D_i, t, \alpha, j) = \frac{\rho(t) \cdot \alpha^{\tau(t)j}}{\sum_{k=1}^n \rho(t_k) \cdot \alpha^{\tau(t_k)j}},$$

where  $\{t_1, t_2, \dots, t_n\}$  is the set of tip nodes of  $D_i$ .

The interpretation of  $P(D_i, t, \alpha, j)$  is the probability of a tip node  $t$  in the navigation tree  $D_i$ . We note that when  $\alpha = 1$  then this probability of  $t$  is purely proportional to the the score of the trail it induces.

6.  $extend(D_i, t)$  returns a navigation tree resulting from the extension of  $D_i$  with tip node  $t$ .
7.  $select(D_i, \alpha, j)$ , where  $\alpha$  is either 1 or  $df$  and  $j$  denotes either an exploration or convergence step, returns a tip of  $D_i$  chosen by a random device operating according to the probability distribution function  $P(D_i, t, \alpha, j)$ .
8.  $best(D_i)$  returns the trail with the highest score from the set of trails induced by the set of tip nodes of  $D_i$ . (If there is more than one highest scoring trail choose the shorter one, otherwise choose any one of them uniformly at random.)
9.  $overall\_best(\{T_1, T_2, \dots, T_M\})$ , where  $\{T_1, T_2, \dots, T_M\}$  is a set of  $M$  trails, returns the highest scoring trail from this set; we call this trail the *best trail*. (If there is more than one highest scoring trail choose the shorter one, otherwise choose any one of them uniformly at random.)

The Best Trail algorithm, whose pseudocode is given in Algorithm 2, takes  $K \geq 1$  starting URLs,  $U_1, U_2, \dots, U_K$ , and a parameter  $M \geq 1$ , which specifies the number of repetitions of the algorithm for each input URL. It outputs a set of  $K$  trails  $\{B_1, B_2, \dots, B_K\}$ , one for each input URL.

**Algorithm 2 (Best\_Trail( $\{U_1, U_2, \dots, U_K\}, M$ ))**

```

1. begin
2.   for  $k = 1$  to  $K$  do
3.     for  $i = 1$  to  $M$  do
4.        $D_i \leftarrow \{U_k\}$ ;
5.       for  $j = 1$  to  $I_{\text{explore}}$  do
6.          $t \leftarrow select(D_i, 1, j)$ ;
7.          $D_i \leftarrow extend(D_i, t)$ ;
8.       end for
9.       for  $j = 1$  to  $I_{\text{converge}}$  do
10.         $t \leftarrow select(D_i, df, j)$ ;
11.         $D_i \leftarrow extend(D_i, t)$ ;
12.      end for
13.       $T_i \leftarrow best(D_i)$ ;
14.    end for
15.     $B_k \leftarrow overall\_best(\{T_1, T_2, \dots, T_M\})$ ;
16.  end for
17.  return  $\{B_1, B_2, \dots, B_K\}$ ;
18. end.
    
```

The algorithm has a main outer for loop starting at line 2 and ending at line 16, which computes the best trail for each one of the  $K$  input URLs. The first inner for loop starting at line 3 and ending at line 14 recomputes the best trail  $M$  times, given the starting URL  $U_k$ . The overall best trail over the  $M$  iterations with the same starting

URL is chosen at line 15 of the algorithm. We note that because of the stochastic nature of the algorithm, we may get different trails  $T_i$  at line 13 of the algorithm from two separate iterations of the for loop starting at line 3 and ending at line 14. The algorithm has two further inner for loops, the first one starting at line 5 and ending at line 8 comprises the exploration stage of the algorithm, and the second one starting at line 9 and ending at line 12 comprises the convergence stage of the algorithm. Finally, the set of  $K$  best trails for the set of  $K$  input URLs is returned at line 17 of the algorithm.

One important issue is removing redundancy from the output trails to increase their relevance to users viewing them. We consider a trail to be redundant if all the pages in the trail are contained in another more relevant trail. Within a trail, we consider a page to be redundant if either it is not relevant to the user query or if the page is duplicated previously in the trail, and removing the page leaves a valid trail with respect to the topology of the Web graph.

Optimisation of the algorithm and implementation issues are discussed in [70]. Therein we also report on experiments we have run to test the behaviour of the algorithm and discuss how to tune its various parameters. The application of the Best Trail algorithm for keyword search with a relational database is discussed in [71].

We now briefly describe a navigation system we have been developing that uses the Best Trail algorithm to construct trails that are relevant to a user's query. The trails are presented to the user in a treelike structure with which he or she can interact. This is in sharp contrast to a search engine, which merely outputs a list of pages that are relevant to the user query without addressing the problem of which trail the user should follow.

The navigation system obtains the preferred trails for navigation, given a user query, from the *navigation engine* and requests pages for display from the Web site via a proxy. The navigation engine consists of two main modules: the information retrieval module, and the best trail module. The information retrieval module does conventional information retrieval over Web pages combined with using the potential gain metric to determine starting points for navigation. The best trail module implements the Best Trail algorithm to compute the preferred trails for navigation given the input query; see [70] for more details.

The main mechanism of the user interface is the *navigation tree window*, which displays the preferred trails given the user query, organised in the form of a tree structure with the trails being ranked from the most preferred, according to their score. The user interacting with the navigation tree window can select any Web page on one of the preferred trails by clicking the corresponding link and thus causing the page to be displayed in the browser window. Putting the cursor over a link in the navigation tree window will cause a small window to popup, displaying the summary of the destination Web page. The mechanisms of the user interface provide the user with guidance and context throughout a navigation session. The user interface can be embodied within a Web site as a navigation mechanism complementing or replacing a Web site search engine. A screen shot for the query "neural networks" is shown in

Fig. 4; an alternative interface which displays trails in graphical form is described in [72]. The trails clearly indicate that both the BSc and MSc courses support neural network options and that Chris is involved in teaching the subject and doing research in this area. (We note that the single page trail, whose title is “Chris’s Home Page”, is in fact a Web page about neural network application projects, and is thus a case where the HTML title was not used effectively. Similarly, the page on the two-page trail with a link to the PDF file nn\_prnt.pdf, whose title is also “Chris’s Home Page”, is in fact a Web page about understanding neuronal coding.)

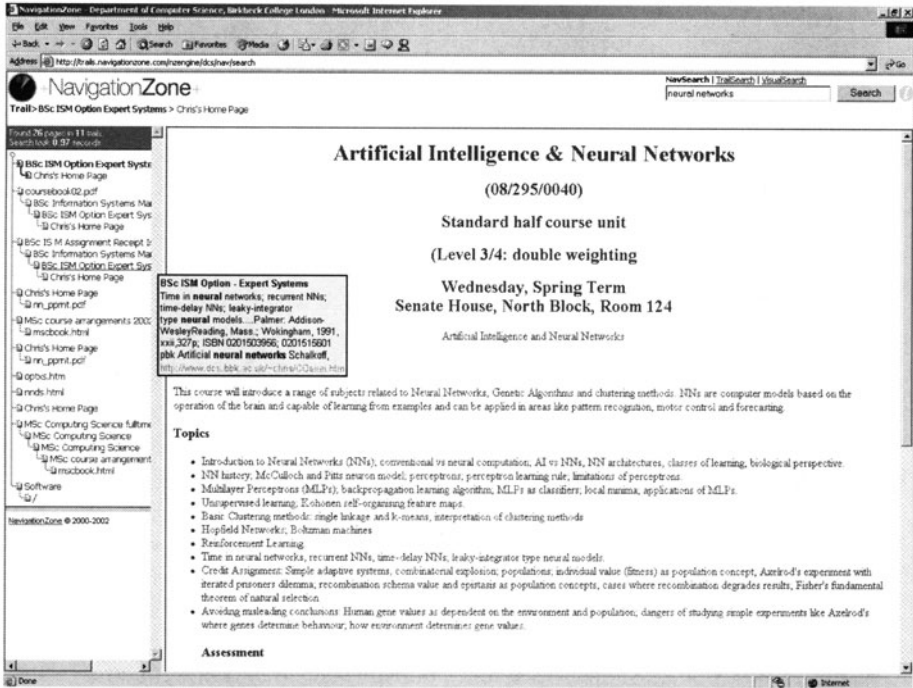


Fig. 4. Navigation engine results for the query “neural networks”

We conclude this section by mentioning a usability study we carried out to test whether our navigation system enhances users’ search experiences [46]. Our results show the potential of navigational assistance in search tools, since overall, users of the navigation system employed fewer clicks and took less time in completing their tasks than those using a conventional search engine. One reason for these positive results may be that users of the navigation system did not have to click on the back button as much as users of a conventional search engine, but instead could use the navigation tree window.

## 8 Trail Records

Here we present a model for analysing the record of trails that emerge from either an individual user or a group of users through navigation within the Web graph over a period a time. The model is within the area of *Web usage mining* [45], which is concerned with finding patterns in surfers' navigation behaviour.

We define a *Web view* (or a *record of trails*) as a collection of trails that are the result of user navigation sessions over a period of time. Thus a Web view is a subgraph of the Web graph induced by a collection of emergent trails. We limit the trails in a Web view by two threshold parameters, as follows:

1. *support*  $\alpha \in [0, 1)$ ; accept into the Web view only trails whose initial probability is greater than  $\alpha$
2. *confidence*  $\beta \in [0, 1)$ ; accept into the Web view only trails whose product of transition probabilities is greater than  $\beta$

Alternatively, we accept into the Web view only trails whose overall probability is above some *cut-point*  $\lambda \in [0, 1)$ , with  $\lambda \geq \alpha \cdot \beta$ .

Let  $\mathcal{M}$  be an ergodic Markov chain modelling user's navigation behaviour within the Web graph. Then a Web view over  $\mathcal{M}$  constrained by  $\lambda$  is the set of all trails  $T$  in  $\mathcal{M}$  such that the probability of  $T$  is greater than  $\lambda$ . An alternative formalisation of a Web view separating the support and confidence thresholds can also be given.

We now describe a technique for constructing a Web view that is concerned with finding frequent user behaviour patterns. In  $\mathcal{M}$  the high probability trails, i.e. those having probability above the cut-point, correspond to the user's preferred trails. We assume that we have at our disposal Web log data; for example, collected from the user's browser or from server logs, which make it is possible to infer user navigation sessions. (The log data could be collected for a group of users rather than a single user if we are interested in collaborative trails rather than personalised trails.) It is customary to define a navigation session as a sequence of page visits (i.e. URL requests) by the user where no two consecutive visits are separated by more than a prescribed amount of time, which is normally not more than half an hour.

When sufficient such log data is available we preprocess this data into a collection of trails, each trail being represented as a sequence of URLs. Moreover, we assume that the start and end URLs of all trails correspond to the user's home page. We note that a trail may appear more than once in this collection, since the user may follow the same trail on two or more different occasions. We then build an ergodic Markov chain (or equivalently HPA), say  $\mathcal{M}$ , whose initial probabilities correspond to the frequency the user visited a page present in any one of the input trails, and whose transition probabilities correspond to the frequency that a link was followed in any one of the input trails. We observe that the states of  $\mathcal{M}$  are the pages the user visited and the topology of  $\mathcal{M}$ , i.e. its underlying graph, is induced by the links the user followed. In constructing  $\mathcal{M}$  we have implicitly assumed that when the user chooses a link to follow he or she does not base his or her decision on the previous pages

visited during the navigation session. That is, we have assumed that  $\mathcal{M}$  is a first-order Markov chain. This assumption can be relaxed so that  $N$  (with  $N \geq 1$ ) previous pages including the current one are taken into account; the case with  $N = 1$  is the first-order case when the user bases his or her decision only on the page currently being browsed. The parameter  $N$  is called the *history depth*.

Given a history depth  $N > 1$ , a higher-order Markov chain can be reduced to a first-order Markov chain by aggregating states. The drawback of such a higher-order Markov chain is the increase in the number of states, which is expected to be  $n \cdot b^{(N-1)}$ , where  $n$  is the number of states in the first-order Markov chain and  $b$  is the average number of out-links embedded in a page. Thus there is a trade-off between the history depth and the complexity of the Markov chain measured by its number of states. The decision on whether the gain in accuracy by adopting a higher-order Markov chain is significant can be aided by statistical techniques [15]. (Another approach we are looking at, which increases the history depth yet maintains a complexity as low as possible, is to use variable-order Markov chains [58] or dynamic Markov modelling [19]; see [41].)

Once the HPA  $\mathcal{M}$  has been constructed from the collection of trails, which have been preprocessed from the log data, we employ a Depth-First Search (DFS) to find all the trails in  $\mathcal{M}$  starting from the user's home page and having probability above the cut-point  $\lambda$ . We have run extensive experiments with synthetic and real data to test the performance of the DFS algorithm [8]. It transpires that for a given cut-point there is a strong linear correlation between the size of  $\mathcal{M}$ , measured by its number of states, and the running time of the algorithm, measured by the number of links it traverses. Moreover, for a given cut-point, the number of mined trails increases linearly with the size of  $\mathcal{M}$ . On the other hand, the number of mined trails increases exponentially with the decrease in the cut-point.

The DFS algorithm has two main drawbacks. First, since it is an exhaustive search it will, in general, return too many trails. Second, if we increase the cut-point to reduce the number of trails, then on average the returned trails become short, and therefore may not be very interesting. These observations have led us to develop two heuristics for mining high quality trails. Our first approach [9], which we call the *fine-grained* heuristic, limits the numbers of trails returned via a stopping parameter, which is between zero and one, that determines an upper bound on the sum of probabilities of the returned trails. The method used to implement this heuristic is to explore trails whose probability is above the cut-point one by one and in decreasing order of probability. When the stopping parameter is zero then the fine-grained heuristic reduces to the DFS algorithm, and as the parameter gets closer to one less trails are returned. Our initial results show that for a given cut-point the number of trails decreases almost linearly with the increase in the stopping parameter, indicating that the stopping parameter provides good control over the number of trails. Our second approach [10], which we call the *inverse-fisheye* heuristic, is a method of obtaining longer trails while controlling their number. This is obtained by having a dynamic cut-point which is high at the initial stage of the exploration in order to limit the number of trails, and decreases in subsequent stages in order to allow further exploration of



the selected trails. The user specifies a maximum exploration depth, which limits the length of the trails returned. Our initial results show that if the initial cut-point is not too low and the decrease in the cut-point at each step is gradual then we can reduce the number of trails while increasing their average length.

A different approach has been put forward by Schechter et al. [60] with the aim of using Web log data to predict the next URL to be requested from a user. Their algorithm essentially constructs a *suffix tree* [3] generated from user trails within a navigation session inferred from the log data. A suffix of a trail is added to the tree only if the occurrence count of the predecessor node of the suffix is greater than a predefined threshold. An algorithm is devised that finds the maximal prefix of a trail in the suffix tree that matches the current user trail, and then uses the found trail to predict the next URL as the next node in the tree that is on this trail. Using our example Web graph shown in Fig. 2, a user trail might correspond to

SCSIS  $\rightarrow$  Research  $\rightarrow$  Students,

matching the prefix of two trails in the suffix tree having Azy and Kevin as their next node. The algorithm will predict either Azy or Kevin as the next page the user will browse, depending on which one was traversed more frequently by the user.

We have extended our model to include physical spaces as well as virtual Web spaces [42]. We now distinguish between a *trail*, which may be through a physical space such as a museum exhibit, and a *trail record*, which is a hypertextual trail providing an account of the user navigation session, be it physical or virtual. Trail records thus provide us with a model of users' actions, which can be viewed as a spatial/temporal account of their activities. Such records of a physical experience act as a memory aid that can be expanded by further experiences and additional content. Moreover, using the data mining techniques we described above, user navigation patterns through the space under consideration can be inferred.

We close this section with a brief discussion of a stochastic model of users surfing the Web [28, 37]. In this model each page a user visits has a value and the user browsing a page has to decide if to continue surfing by clicking on a link or to stop surfing. In Huberman et al. [28] the user will continue surfing if the expected cost, to the user, of continuing is smaller than the expected gain obtained from future information in pages which will be browsed. Analysis based on this model leads to a power-law distribution, where the probability of surfing by following  $L$  links is proportional to  $L^{-3/2}$ . Huberman's model does not take into account the topology of the Web graph being navigated. In Levene et al. [37] the user is assumed to be navigating within a Markov chain representing the Web graph, where longer trails are less probable than shorter trails. Again a power-law distribution, which is proportional to the length of the trail being navigated, is derived for the probability of surfing.

## 9 Navigation in the Mobile Web

The ability to connect to the internet through mobile devices such as mobile phones and handheld and portable computing devices means that we can be connected "any-

time” and “anywhere”. The limitations of mobile devices in terms of screen size, computing power and lack of traditional input devices, such as keyboard and mouse, means that alternative input modalities such as pen and voice will become prevalent, and innovative software solutions such as voice recognition, handwriting recognition and predictive text systems will be necessary. Information needs that do not require complex and lengthy navigation, such as browsing news headlines, addresses and train schedules, can readily be supported on mobile devices. Other information needs which are more navigational, such as information gathering on a particular topic are poorly suited for mobile devices [62]. The issues relating to the user interface design on mobile devices to support search are discussed in the chapter by Smyth and Cotter.

As location sensing technologies are already widely available [27], *location-aware services*, which focus the application to the physical location of a user, can play an important role in narrowing down the information need a mobile user has. For example, technologies such as GPS can assist users in physical navigation, such as helping them to find a local restaurant serving their favourite cuisine. We note that we can also add the time dimension to mobile services to further narrow down the information need; in the local restaurant example, the time of day will obviously have an effect on the answer.

An interesting specialisation of a common navigation tool is that of *dynamic bookmarks* [22], where bookmarked pages may vary from location to location. As opposed to storing the URL of the bookmarked page, a dynamic bookmark is associated with a name and a set of attributes such as the type of service required. One idea that has to be further investigated is the creation of *dynamic authored trails*, which provide the user with a specific set of information needs. For example, the trail

News Headlines → Top Shares → Directions to Hotel → Nearest Restaurant

may be useful on a business trip, where each information need is only instantiated at the instruction of the user within a time and location context.

We briefly mention two recent attempts to improve navigation from mobile devices. Anderson et al. [2] developed an adaptive algorithm to improve Web navigation for mobile devices by adding shortcut links to Web pages, thus allowing the user to reach a destination page as quickly as possible. Their algorithm for creating high-quality shortcuts is based on the user’s navigation history. Smyth and Cotter [64] investigate the navigation problem within mobile portals, where personalisation techniques can reduce the click-distance to relevant information. By collecting the user’s past clicks on menu items, conditional probabilities can be computed and more probable paths suggested to the user.

## 10 Navigation in Social Networks

Viewing the Web as a social network has resulted in novel techniques to enhance Web search and navigation [34]. In this context several researchers have gathered evidence

that the Web is a *power-law network* in the sense that its in-degree distribution (and to a lesser extent its out-degree distribution) follows a power law [12]. Moreover, the Web obeys the *small-world* property of having a short average distances between any two connected pages; this distance was shown to be roughly 16 clicks [12].

The fact that short paths exist between two pages does not imply that they are computationally easy to find. Kleinberg [33] investigated this problem in the context of a two-dimensional lattice structure, where nodes have short-range links to their immediate neighbours in the lattice and long-range links to more distant nodes according to a parameter that determines the probability of a link between two nodes as a function of their lattice distance. He found that a decentralised algorithm, in which the only information known to the searcher of a short path is purely local, exists if and only if the parameter's value is two, i.e. long-range links follow an inverse-square distribution. The algorithm is "greedy" in the sense that it chooses a link that brings it as close as possible to the target.

Adamic et al. [1] consider a more realistic scenario, where no global information about the position of the target node is available to the searcher, and where the topology is not restricted to a lattice. They show that random walks in power-law networks naturally lead to nodes with a high in-degree but that intentionally following nodes with a high in-degree performs even better. From their analysis they conclude that local search strategies in power-law graphs scale sublinearly with the size of the graph. Similar results were shown by Kim et al. [31], who formulated the problem in terms of finding short paths between two nodes in the network.

Watts et al. [69] consider the property of *searchability* in social networks, where searchability is defined as the property of being able to find a target quickly. We recast their model in terms of locating pages in the Web graph. In this model Web pages are the individuals in the social network, and their out-going links are their network neighbours. Each Web page can have several semantic categories attached to it; these are the set of *identities* of the page. For example, the School's home page may be categorised both in terms of its subject area, i.e. Computer Science, and in terms of its location within the University of London, implying that the page belongs to two *groups* of Web pages according to its dual identity. We assume it possible to measure similarity between pairs of pages within an ontology, for example, within a category of the Open Directory structure. The similarity between two pages within a category can be defined as the least number of levels needed to reach a common ancestor category; other similarity measures, say involving textual similarity, are possible. The *social distance* between two pages, each having several identities, is defined as the smallest distance over all common identities that the two pages may have. So, for example, the distance between the School of Computer Science and a noncomputing school within the University of London may be one, while the distance between the School of Computer Science and another school of computing outside the University of London may be greater than one due to different research area specialisations. Under some distributional assumptions, which enable simulation of the model, Watts et al. [69] show that using only local knowledge of the neighbours of a page within the network and knowledge of the set of identities of the target page, the target can be

found quickly. The algorithm to find the target is a “greedy” one similar to Kleinberg’s algorithm [33], which chooses to follow the link to a neighbour that is closest to the target in terms of social distance.

This approach seems to reconcile directory-based navigation and link-based navigation on the Web, where users navigate within the Web topology, choosing links according to their semantic proximity to the goal. It also has some similarity with the Best Trail algorithm, presented above, where navigation is link-based but the choice of link is made according to the information need of the user.

## 11 Open Problems

We close this chapter with a list of several open research problems that warrant further investigation.

- One important issue that is not well understood is how to evaluate navigation trails. In the Information Retrieval community there is a strong tradition of evaluation in terms of precision and recall [4], but it is not clear how these notions carry over to the realm of trails.
- It would be interesting to compare the potential gain and gain rank metrics to other Web metrics such as PageRank and investigate whether it is beneficial for them to be combined in some manner.
- Another issue that is still open is how to incorporate personalisation and collaboration within the context of our model of trail records.
- The Best Trail is essentially a probabilistic best first algorithm. It would be worth investigating whether it could be applied within the artificial intelligence domain.
- Designing novel user interfaces for trail-based systems is another area that is under developed, yet highly important.

## References

1. L.A. Adamic, R.M. Lukose, A.R. Puniyani, and B.A. Huberman. Search in power-law networks. *Physical Review E*, 64:046135, 2001.
2. C.R. Anderson, P. Domingos, and D.S. Weld. Adaptive Web navigation for wireless devices. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 879–884, Seattle, Washington, 2001.
3. A. Apostolico. The myriad virtues of subword trees. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume F12 of *NATO ASI Series*, pages 85–96. Springer, Berlin Heidelberg New York, 1985.
4. R.A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press and Addison-Wesley, Reading, Ma., 1999.
5. M.K. Bergman. The deep Web: Surfacing hidden value. White paper, Bright Planet, July 2000.
6. T. Berners-Lee. *Weaving the Web*. Orion Books, London, 1999.

7. M. Bernstein. The bookmark and the compass: Orientation tools for hypertext users. *SIGOIS Bulletin*, 9:34–45, 1988.
8. J. Borges and M. Levene. Data mining of user navigation patterns. In B. Masand and M. Spiliopoulou, editors, *Web Usage Analysis and User Profiling*, Lecture Notes in Artificial Intelligence (LNAI 1836), pages 92–111. Springer, Berlin Heidelberg New York, 2000.
9. J. Borges and M. Levene. A fine grained heuristic to capture Web navigation patterns. *SIGKDD Explorations*, 2:40–50, 2000.
10. J. Borges and M. Levene. A heuristic to capture longer user Web navigation patterns. In *Proceedings of International Conference on Electronic Commerce and Web Technologies (EC-Web)*, pages 155–164, Greenwich, U.K., 2000.
11. A. Broder. A taxonomy of Web search. *SIGIR Forum*, 36, Fall 2002.
12. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, A. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the Web. *Computer Networks*, 33:309–320, 2000.
13. F. Buckley and F. Harary. *Distance in Graphs*. Addison-Wesley, Redwood City, CA, 1990.
14. V. Bush. As we may think. *Atlantic Monthly*, 176:101–108, 1945.
15. C. Chatfield. Statistical inference regarding Markov chain models. *Applied Statistics*, 22:7–20, 1973.
16. E.H. Chi, P. Pirolli, and J.E. Pitkow. The scent of a site: A system for analyzing and predicting information scent, usgae, and usability of a Web site. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, pages 161–168, The Hague, Amsterdam, 2000.
17. A. Cockburn and B. McKenzie. What do Web users do? An empirical analysis of Web use. *International Journal of Human-Computer Studies*, 54:903–922, 2001.
18. A. Cockburn, B. McKenzie, and M. JasonSmith. Pushing back: evaluating a new behaviour for the back and forward buttons in Web browsers. *International Journal of Human-Computer Studies*, 57:397–414, 2002.
19. G.V. Cormack and R.N.S. Horspool. Data compression using dynamic Markov modelling. *The Computer Journal*, 30:541–550, 1987.
20. P.M.E. De Bra and R.D.J. Post. Searching for arbitrary information in the WWW: The fish-search for Mosaic. In *Proceedings of International World Wide Web Conference*, Chicago, IL, 1994.
21. D. Dhyani, W.K. Ng, and S.S. Bhowmick. A survey of Web metrics. *ACM Computing Surveys*, 34:469–503, 2002.
22. S. Duri, A. Cole, J. Munson, and J. Christensen. An approach to providing a seamless end-user experience for location-aware applications. In *Proceedings of the first international workshop on Mobile commerce*, pages 20–25, Rome, 2001.
23. D.C. Englebart and W.K. English. A research center for augmenting human intellect. In *Proceedings of AFIPS Fall Joint Conference*, pages 395–3410, San Francisco, CA, 1968.
24. G.W. Furnas. Effective view navigation. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, pages 367–374, Atlanta, GA, 1997.
25. M.A. Hearst. Next generation Web search: Setting our sites. *Bulletin of the Technical Committee on Data Engineering*, 23:38–48, 2000.
26. M. Hersovici, M. Jacovi, Y.S. Maarek, D. Pelleg, M. Shtalheim, and S. Ur. The shark-search algorithm - An application: Tailored Web site mapping. In *Proceedings of International World Wide Web Conference*, pages 317–326, Brisbane, 1998.
27. J. Hightower and G. Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 34:57–66, 2001.

28. B.A. Huberman, P.L.T. Pirolli, J.E. Pitkow, and R.M. Lukose. Strong regularities in World Wide Web surfing. *Science*, 280:95–97, 1998.
29. T. Joachims, D. Freitag, and T. Mitchell. WebWatcher: A tour guide for the World Wide Web. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 770–775, Nagoya, Japan, 1997.
30. J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. D. Van Nostrand, Princeton, NJ, 1960.
31. B.J. Kim, C.N. Yoon, S.K. Han, and H. Jeong. Path finding strategies in scale-free networks. *Physical Review E*, 65:027103, 2002.
32. H. Kim and S.C. Hirtle. Spatial metaphors and disorientation in hypertext browsing. *Behaviour and Information Technology*, 14:239–250, 1995.
33. J. Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000.
34. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The Web and social networks. *IEEE Computer*, 35:32–36, 2002.
35. J. Lamping and R. Rao. The hyperbolic browser: A focus + context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, 7:33–55, 1996.
36. S. Lawrence and C.L. Giles. Accessibility of information on the Web. *Nature*, 400:107–109, 1999.
37. M. Levene, J. Borges, and G. Loizou. Zipf’s law for Web surfers. *Knowledge and Information Systems*, 3:120–129, 2001.
38. M. Levene and G. Loizou. Navigation in hypertext is easy only sometimes. *SIAM Journal on Computing*, 29:728–760, 1999.
39. M. Levene and G. Loizou. A probabilistic approach to navigation in hypertext. *Information Sciences*, 114:165–186, 1999.
40. M. Levene and G. Loizou. Kemeny’s constant and the random surfer. *American Mathematical Monthly*, 109:741–745, 2002.
41. M. Levene and G. Loizou. Computing the entropy of user navigation in the Web. *International Journal of Information Technology and Decision Making*, 2:459–476, 2003.
42. M. Levene and D. Peterson. Trail records and ampliative learning. Research Report BBKCS-02-10, School of Information Systems and Computer Science, Birkbeck University of London, October 2002.
43. G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7:76–80, 2003.
44. M. Marchiori. The quest for correct information on the Web: Hyper search engines. In *Proceedings of International World Wide Web Conference*, pages 265–276, Santa Clara, CA, 1997.
45. B. Masand and M. Spiliopoulou, editors. *Web Usage Analysis and User Profiling*, volume 1836 of *Lecture Notes in Artificial Intelligence*. Springer, Berlin Heidelberg New York, 2000.
46. M. Mat-Hassan and M. Levene. Can navigational assistance improve search experience: A user study. *First Monday*, 6(9), 2001.
47. F. Menczer and R.K. Belew. Adaptive retrieval agents: Internalizing local context and scaling up to the Web. *Machine Learning*, 39:203–242, 2000.
48. S. Mukherjea and J.D. Foley. Showing the context of nodes in the World-Wide Web. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, Denver, Colorado, 1995. Short paper.
49. B.H. Murray and A. Moore. Sizing the internet. White paper, Cyveillance, July 2000.

50. T. Nelson. Xanalogical structure, needed now more than ever: parallel documents, deep links to content, deep versioning, and deep re-use. *ACM Computing Surveys*, 31:1–32, 1999.
51. J. Nielsen. *Hypertext and Hypermedia*. Academic Press, Boston, Ma., 1990.
52. J. Nielsen. *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, Indianapolis, Indiana, 2000.
53. J.M. Nyce and P. Kahn, editors. *From Memex to Hypertext: Vannevar Bush and the Mind's Machine*. Academic Press, San Diego, CA, 1991.
54. T. Oren. Memex: Getting back on the trail. In J.M. Nyce and P. Kahn, editors, *From Memex to Hypertext: Vannevar Bush and the Mind's Machine*, pages 319–338. Academic Press, San Diego, CA, 1991.
55. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the Web. Working paper, Department of Computer Science, Stanford University, 1998.
56. P. Pirolli, J.E. Pitkow, and R. Rao. Silk from a sow's ear: Extracting usable structures from the Web. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, pages 118–125, Vancouver, 1996.
57. E. Rivlin, R. Botafogo, and B. Shneiderman. Navigating in hyperspace: Designing a structure-based toolbox. *Communications of the ACM*, 37:87–96, 1994.
58. D. Ron, Y. Singer, and N. Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25:117–149, 1996.
59. G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY, 1983.
60. S. Schechter, M. Krishnan, and M.D. Smith. Using path profiles to predict HTTP requests. *Computer Networks and ISDN Systems*, 30:457–467, 1998.
61. F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34:1–47, 2002.
62. A.J. Sellen and R. Murphy. The future of the mobile internet: Lessons from looking at Web use. Technical Report HPL-2002-230, Information Infrastructure Laboratory, HP Laboratories, Bristol, August 2002.
63. F. Shipman, R. Furuta, D. Brenner, C. Chung, and H. Hsieh. Guided paths through Web-based collections: Design, experiences, and adaptations. *Journal of the American Society for Information Science*, 51:260–272, 2000.
64. B. Smyth and P. Cotter. Personalized adaptive navigation for mobile portals. In *ECAI2002. Proceedings of the 15th European Conference on Artificial Intelligence*, pages 608–612, Lyon, France, 2002.
65. R.S. Sutton and A.G. Barto. *Reinforcement Learning*. MIT Press, Cambridge, Ma., 1998.
66. L. Tauscher and S. Greenberg. How people revisit Web pages: Empirical findings and implications for the design of history systems. *International Journal of Human-Computer Studies*, 47:97–137, 1997.
67. K. Tochtermann and G. Dittrich. Fishing for clarity in hyperdocuments with enhanced fisheye-views. In *Proceedings of ACM Conference on Hypertext*, pages 212–221, Milano, Italy, 1992.
68. R.H. Trigg. From trailblazing to guided tours: The legacy of Vannevar Bush's vision of hypertext use. In J.M. Nyce and P. Kahn, editors, *From Memex to Hypertext: Vannevar Bush and the Mind's Machine*, pages 353–367. Academic Press, San Diego, CA, 1991.
69. D.J. Watts, P.S. Dodds, and M.E.J. Newman. Identity and search in social networks. *Science*, 296:1302–1305, 2002.

70. R. Wheeldon and M. Levene. The best trail algorithm for adaptive navigation in the World-Wide-Web. In *Proceedings of 1st Latin American Web Congress*, Santiago, Chile, November 2003.
71. R. Wheeldon, M. Levene, and K. Keenoy. Search and navigation in relational databases. *Computing Research Repository*, cs.DB/0307073, July 2002.
72. R. Wheeldon, M. Levene, and N. Zin. Autodoc: A search and navigation tool for Web-based program documentation. In *Poster Proceedings of International World Wide Web Conference*, Hawaii, 2002.



---

# Crawling the Web

Gautam Pant<sup>1</sup>, Padmini Srinivasan<sup>1,2</sup>, and Filippo Menczer<sup>3</sup>

<sup>1</sup> Department of Management Sciences

<sup>2</sup> School of Library and Information Science

The University of Iowa, Iowa City IA 52242, USA

{gautam-pant,padmini-srinivasan}@uiowa.edu

<sup>3</sup> School of Informatics

Indiana University, Bloomington, IN 47408, USA

fil@indiana.edu

**Summary.** The large size and the dynamic nature of the Web make it necessary to continually maintain Web based information retrieval systems. Crawlers facilitate this process by following hyperlinks in Web pages to automatically download new and updated Web pages. While some systems rely on crawlers that exhaustively crawl the Web, others incorporate “focus” within their crawlers to harvest application- or topic-specific collections. In this chapter we discuss the basic issues related to developing an infrastructure for crawlers. This is followed by a review of several topical crawling algorithms, and evaluation metrics that may be used to judge their performance. Given that many innovative applications of Web crawling are still being invented, we briefly discuss some that have already been developed.

## 1 Introduction

Web *crawlers* are programs that exploit the graph structure of the Web to move from page to page. In their infancy such programs were also called wanderers, robots, spiders, fish, and worms, words that are quite evocative of Web imagery. It may be observed that the noun “crawler” is not indicative of the speed of these programs, as they can be considerably fast. In our own experience, we have been able to crawl up to tens of thousands of pages within a few minutes while consuming a small fraction of the available bandwidth.<sup>4</sup>

From the beginning, a key motivation for designing Web crawlers has been to retrieve Web pages and add them or their representations to a local repository. Such a repository may then serve particular application needs such as those of a Web search engine. In its simplest form a crawler starts from a *seed* page and then uses the external links within it to attend to other pages. The process repeats with the new pages

---

<sup>4</sup> We used a Pentium 4 workstation with an Internet2 connection.

offering more external links to follow, until a sufficient number of pages are identified or some higher-level objective is reached. Behind this simple description lies a host of issues related to network connections, spider traps, canonicalizing URLs, parsing HTML pages, and the ethics of dealing with remote Web servers. In fact, a current generation Web crawler can be one of the most sophisticated yet fragile parts [5] of the application in which it is embedded.

Were the Web a static collection of pages we would have little long-term use for crawling. Once all the pages had been fetched to a repository (like a search engine's database), there would be no further need for crawling. However, the Web is a dynamic entity with subspaces evolving at differing and often rapid rates. Hence there is a continual need for crawlers to help applications stay current as new pages are added and old ones are deleted, moved or modified.

General-purpose search engines serving as entry points to Web pages strive for coverage that is as broad as possible. They use Web crawlers to maintain their index databases [3], amortizing the cost of crawling and indexing over the millions of queries received by them. These crawlers are blind and exhaustive in their approach, with comprehensiveness as their major goal. In contrast, crawlers can be selective about the pages they fetch and are then referred to as *preferential* or heuristic-based crawlers [10, 6]. These may be used for building focused repositories, automating resource discovery, and facilitating software agents. There is a vast literature on preferential crawling applications including [15, 9, 31, 20, 26, 3]. Preferential crawlers built to retrieve pages within a certain topic are called *topical* or *focused* crawlers. Synergism between search engines and topical crawlers is certainly possible, with the latter taking on the specialized responsibility of identifying subspaces relevant to particular communities of users. Techniques for preferential crawling that focus on improving the "freshness" of a search engine have also been suggested [3].

Although a significant portion of this chapter is devoted to description of crawlers in general, the overall slant, particularly in the latter sections, is toward topical crawlers. There are several dimensions about topical crawlers that make them an exciting object of study. One key question that has motivated much research is: How is crawler selectivity to be achieved? Rich contextual aspects, such as the goals of the parent application, lexical signals within the Web pages, and also features of the graph built from pages already seen, are all reasonable kinds of evidence to exploit. Additionally, crawlers can and often do differ in their mechanisms for using the evidence available to them.

A second major aspect that is important to consider when studying crawlers, especially topical crawlers, is the nature of the crawl task. Crawl characteristics such as queries and/or keywords provided as input criteria to the crawler, user-profiles, and desired properties of the pages to be fetched (similar pages, popular pages, authoritative pages, etc.) can lead to significant differences in crawler design and implementation. The task could be constrained by parameters like the maximum number of pages to be fetched (long crawls versus short crawls) or the available memory. Hence, a crawling task can be viewed as a constrained multiobjective search problem. However, the wide variety of objective functions, coupled with the lack of appropriate knowledge about

the search space, make the problem a hard one. Furthermore, a crawler may have to deal with optimization issues such as local versus global optima [28].

The last key dimension is regarding crawler evaluation strategies necessary to make comparisons and determine circumstances under which one or the other crawlers work best. Comparisons must be fair and be made with an eye toward drawing out statistically significant differences. Not only does this require a sufficient number of crawl runs but also sound methodologies that consider the temporal nature of crawler outputs. Significant challenges in evaluation include the general unavailability of relevant sets for particular topics or queries. Thus evaluation typically relies on defining measures for estimating page importance.

The first part of this chapter presents a crawling infrastructure and within this describes the basic concepts in Web crawling. Following this, we review a number of crawling algorithms that are suggested in the literature. We then discuss current methods to evaluate and compare performance of different crawlers. Finally, we outline the use of Web crawlers in some applications.

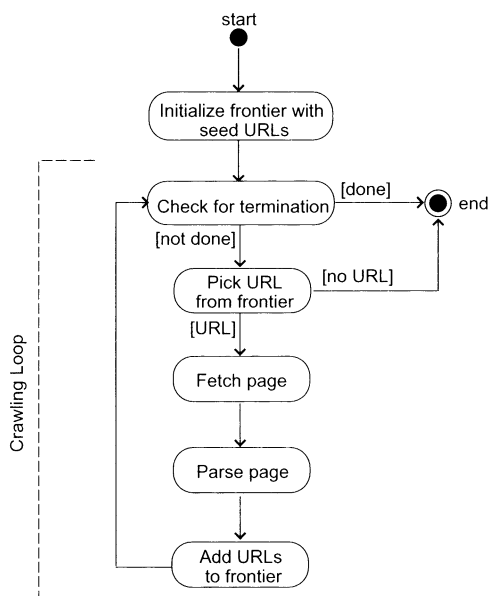
## 2 Building a Crawling Infrastructure

Figure 1 shows the flow of a basic sequential crawler (in Sect. 2.6 we consider multi-threaded crawlers). The crawler maintains a list of unvisited URLs called the *frontier*. The list is initialized with seed URLs, which may be provided by a user or another program. Each *crawling loop* involves picking the next URL to crawl from the frontier, fetching the page corresponding to the URL through HTTP, parsing the retrieved page to extract the URLs and application-specific information, and finally adding the unvisited URLs to the frontier. Before the URLs are added to the frontier they may be assigned a score that represents the estimated benefit of visiting the page corresponding to the URL. The crawling process may be terminated when a certain number of pages have been crawled. If the crawler is ready to crawl another page and the frontier is empty, the situation signals a deadend for the crawler. The crawler has no new page to fetch, and hence it stops.

Crawling can be viewed as a graph search problem. The Web is seen as a large graph with pages at its nodes and hyperlinks as its edges. A crawler starts at a few of the nodes (seeds) and then follows the edges to reach other nodes. The process of fetching a page and extracting the links within it is analogous to expanding a node in graph search. A topical crawler tries to follow edges that are expected to lead to portions of the graph that are relevant to a topic.

### 2.1 Frontier

The frontier is the to-do list of a crawler that contains the URLs of unvisited pages. In graph search terminology the frontier is an *open list* of unexpanded (unvisited) nodes. Although it may be necessary to store the frontier on disk for large-scale crawlers, we will represent the frontier as an in-memory data structure for simplicity. Based on the available memory, one can decide the maximum size of the frontier. Because of the



**Fig. 1.** Flow of a basic sequential crawler

large amount of memory available on PCs today, a frontier size of a 100,000 URLs or more is not exceptional. Given a maximum frontier size we need a mechanism to decide which URLs to ignore when this limit is reached. Note that the frontier can fill rather quickly as pages are crawled. One can expect around 60,000 URLs in the frontier with a crawl of 10,000 pages, assuming an average of about 7 links per page [30].

The frontier may be implemented as a FIFO queue, in which case we have a breadth-first crawler that can be used to blindly crawl the Web. The URL to crawl next comes from the head of the queue, and the new URLs are added to the tail of the queue. Because of the limited size of the frontier, we need to make sure that we do not add duplicate URLs into the frontier. A linear search to find out if a newly extracted URL is already in the frontier is costly. One solution is to allocate some amount of available memory to maintain a separate hash-table (with URL as key) to store each of the frontier URLs for fast lookup. The hash-table must be kept synchronized with the actual frontier. A more time-consuming alternative is to maintain the frontier itself as a hash-table (again with URL as key). This would provide fast lookup for avoiding duplicate URLs. However, each time the crawler needs a URL to crawl, it would need to search and pick the URL with the earliest time stamp (the time when a URL was added to the frontier). If memory is less of an issue than speed, the first solution may be preferred. Once the frontier reaches its maximum size, the breadth-first crawler can add only one unvisited URL from each new page crawled.

If the frontier is implemented as a priority queue we have a preferential crawler, which is also known as a best-first crawler. The priority queue may be a dynamic

array that is always kept sorted by the estimated score of unvisited URLs. At each step, the best URL is picked from the head of the queue. Once the corresponding page is fetched, the URLs are extracted from it and scored based on some heuristic. They are then added to the frontier in such a manner that the order of the priority queue is maintained. We can avoid duplicate URLs in the frontier by keeping a separate hash-table for lookup. Once the frontier's maximum size (MAX) is exceeded, only the best MAX URLs are kept in the frontier.

If the crawler finds the frontier empty when it needs the next URL to crawl, the crawling process comes to a halt. With a large value of MAX and several seed URLs the frontier will rarely reach the empty state.

At times, a crawler may encounter a *spider trap* that leads it to a large number of different URLs that refer to the same page. One way to alleviate this problem is by limiting the number of pages that the crawler accesses from a given domain. The code associated with the frontier can make sure that every consecutive sequence of  $k$  (say 100) URLs, picked by the crawler, contains only one URL from a fully qualified host name (e.g., `www.cnn.com`). As side effects, the crawler is polite by not accessing the same Web site too often [14], and the crawled pages tend to be more diverse.

## 2.2 History and Page Repository

The crawl history is a time-stamped list of URLs that were fetched by the crawler. In effect, it shows the path of the crawler through the Web, starting from the seed pages. A URL entry is made into the history only after fetching the corresponding page. This history may be used for post-crawl analysis and evaluations. For example, we can associate a value with each page on the crawl path and identify significant events (such as the discovery of an excellent resource). While history may be stored occasionally to the disk, it is also maintained as an in-memory data structure. This provides for a fast lookup to check whether a page has been crawled or not. This check is important to avoid revisiting pages and also to avoid adding the URLs of crawled pages to the limited size frontier. For the same reasons it is important to *canonicalize* the URLs (Sect. 2.4) before adding them to the history.

Once a page is fetched, it may be stored/indexed for the master application (such as a search engine). In its simplest form a *page repository* may store the crawled pages as separate files. In that case, each page must map to a unique file name. One way to do this is to map each page's URL to a compact string using some form of hashing function with low probability of collisions (for uniqueness of file names). The resulting hash value is used as the file name. We use the MD5 one-way hashing function that provides a 128-bit hash code for each URL. Implementations of MD5 and other hashing algorithms are readily available in different programming languages (e.g., refer to Java 2 security framework<sup>5</sup>). The 128-bit hash value is then converted into a 32-character hexadecimal equivalent to get the file name. For example, the content of `http://www.uiowa.edu/` is stored into a file named `160766577426e1d01fcb7735091ec584`. This way we have fixed-length file

<sup>5</sup> `http://java.sun.com`

names for URLs of arbitrary size. (Of course, if the application needs to cache only a few thousand pages, one may use a simpler hashing mechanism.) The page repository can also be used to check if a URL has been crawled before by converting it to its 32-character file name and checking for the existence of that file in the repository. In some cases this may render unnecessary the use of an in-memory history data structure.

## 2.3 Fetching

In order to fetch a Web page, we need an HTTP client that sends an HTTP request for a page and reads the response. The client needs to have timeouts to make sure that an unnecessary amount of time is not spent on slow servers or in reading large pages. In fact, we may typically restrict the client to download only the first 10–20KB of the page. The client needs to parse the response headers for status codes and redirections. We may also like to parse and store the last-modified header to determine the age of the document. Error checking and exception handling are important during the page-fetching process since we need to deal with millions of remote servers using the same code. In addition, it may be beneficial to collect statistics on timeouts and status codes for identifying problems or automatically changing timeout values. Modern programming languages such as Java and Perl provide very simple and often multiple programmatic interfaces for fetching pages from the Web. However, one must be careful in using high-level interfaces where it may be harder to find lower-level problems. For example, with Java one may want to use the `java.net.Socket` class to send HTTP requests instead of using the more ready-made `java.net.HttpURLConnection` class.

No discussion about crawling pages from the Web can be complete without talking about the *Robot Exclusion Protocol*. This protocol provides a mechanism for Web server administrators to communicate their file access policies; more specifically to identify files that may not be accessed by a crawler. This is done by keeping a file named `robots.txt` under the root directory of the Web server (such as `http://www.biz.uiowa.edu/robots.txt`). This file provides access policy for different *User-agents* (robots or crawlers). A User-agent value of “\*” denotes a default policy for any crawler that does not match other User-agent values in the file. A number of *Disallow* entries may be provided for a User-agent. Any URL that starts with the value of a Disallow field must not be retrieved by a crawler matching the User-agent. When a crawler wants to retrieve a page from a Web server, it must first fetch the appropriate `robots.txt` file and make sure that the URL to be fetched is not disallowed. More details on this exclusion protocol can be found at <http://www.robotstxt.org/wc/norobots.html>. It is efficient to cache the access policies of a number of servers recently visited by the crawler. This would avoid accessing a `robots.txt` file each time you need to fetch a URL. However, one must make sure that cache entries remain sufficiently fresh.

## 2.4 Parsing

Once a page has been fetched, we need to parse its content to extract information that will feed and possibly guide the future path of the crawler. Parsing may imply simple hyperlink/URL extraction or it may involve the more complex process of tidying up the HTML content in order to analyze the HTML tag tree (Sect. 2.5). Parsing might also involve steps to convert the extracted URL to a canonical form, remove stopwords from the page's content, and stem the remaining words. These components of parsing are described next.

### URL Extraction and Canonicalization

HTML parsers are freely available for many different languages. They provide the functionality to easily identify HTML tags and associated attribute–value pairs in a given HTML document. In order to extract hyperlink URLs from a Web page, we can use these parsers to find anchor tags and grab the values of associated `href` attributes. However, we do need to convert any relative URLs to absolute URLs using the base URL of the page from where they were retrieved.

Different URLs that correspond to the same Web page can be mapped onto a single canonical form. This is important in order to avoid fetching the same page many times. Here are some of the steps used in typical URL canonicalization procedures:

- Convert the protocol and hostname to lowercase. For example, `HTTP://www.UIOWA.edu` is converted to `http://www.uiowa.edu`.
- Remove the “anchor” or “reference” part of the URL. Hence, `http://myspiders.biz.uiowa.edu/faq.html#what` is reduced to `http://myspiders.biz.uiowa.edu/faq.html`.
- Perform URL encoding for some commonly used characters such as “~”. This would prevent the crawler from treating `http://dollar.biz.uiowa.edu/~pant/` as a different URL from `http://dollar.biz.uiowa.edu/%7Epant/`.
- For some URLs, add trailing “/”s. `http://dollar.biz.uiowa.edu` and `http://dollar.biz.uiowa.edu/` must map to the same canonical form. The decision to add a trailing “/” will require heuristics in many cases.
- Use heuristics to recognize default Web pages. File names such as `index.html` or `index.htm` may be removed from the URL with the assumption that they are the default files. If that is true, they would be retrieved by simply using the base URL.
- Remove “..” and its parent directory from the URL path. Therefore, URL path `/%7Epant/BizIntel/Seeds/../../ODPSeeds.dat` is reduced to `/%7Epant/BizIntel/ODPSeeds.dat`.
- Leave the port numbers in the URL unless it is port 80. As an alternative, leave the port numbers in the URL and add port 80 when no port number is specified.

It is important to be consistent while applying canonicalization rules. It is possible that two seemingly opposite rules work equally well (such as that for port numbers)

as long as you apply them consistently across URLs. Other canonicalization rules may be applied based on the application and prior knowledge about some sites (e.g., known mirrors).

As noted earlier spider traps pose a serious problem for a crawler. The “dummy” URLs created by spider traps often become increasingly larger in size. A way to tackle such traps is by limiting the URL sizes to, say, 128 or 256 characters.

## Stoplisting and Stemming

When parsing a Web page to extract content information or in order to score new URLs suggested by the page, it is often helpful to remove commonly used words or *stopwords*<sup>6</sup> such as “it” and “can”. This process of removing stopwords from text is called *stoplisting*. Note that the Dialog<sup>7</sup> system recognizes no more than nine words (“an,” “and,” “by,” “for,” “from,” “of,” “the,” “to,” and “with”) as the stopwords. In addition to stoplisting, one may also stem the words found in the page. The *stemming* process normalizes words by conflating a number of morphologically similar words to a single root form or stem. For example, “connect,” “connected,” and “connection” are all reduced to “connect.” Implementations of the commonly used Porter stemming algorithm [29] are easily available in many programming languages. One of the authors has experienced cases in the biomedical domain where stemming reduced the precision of the crawling results.

## 2.5 HTML tag tree

Crawlers may assess the value of a URL or a content word by examining the HTML tag context in which it resides. For this, a crawler may need to utilize the tag tree or Document Object Model (DOM) structure of the HTML page [8, 24, 27]. Figure 2 shows a tag tree corresponding to an HTML source. The `<html>` tag forms the root of the tree, and various tags and texts form nodes of the tree. Unfortunately, many Web pages contain badly written HTML. For example, a start tag may not have an end tag (it may not be required by the HTML specification), or the tags may not be properly nested. In many cases, the `<html>` tag or the `<body>` tag is altogether missing from the HTML page. Thus structure-based criteria often require the prior step of converting a “dirty” HTML document into a well-formed one, a process that is called *tidying* an HTML page.<sup>8</sup> This includes both the insertion of missing tags and the reordering of tags in the page. Tidying an HTML page is necessary for mapping the content of a page onto a tree structure with integrity, where each node has a single parent. Hence, it is an essential precursor to analyzing an HTML page as a tag tree. Note that analyzing the DOM structure is only necessary if the topical crawler intends to use the HTML document structure in a nontrivial manner. For example, if

<sup>6</sup> for an example list of stopwords refer to <http://www.dcs.gla.ac.uk/edom/ir-resources/linguisticutils/stop-words>

<sup>7</sup> <http://www.dialog.com>

<sup>8</sup> <http://www.w3.org/People/Raggett/tidy/>



the crawler only needs the links within a page, and the text or portions of the text in the page, one can use simpler HTML parsers. Such parsers are also readily available in many languages.

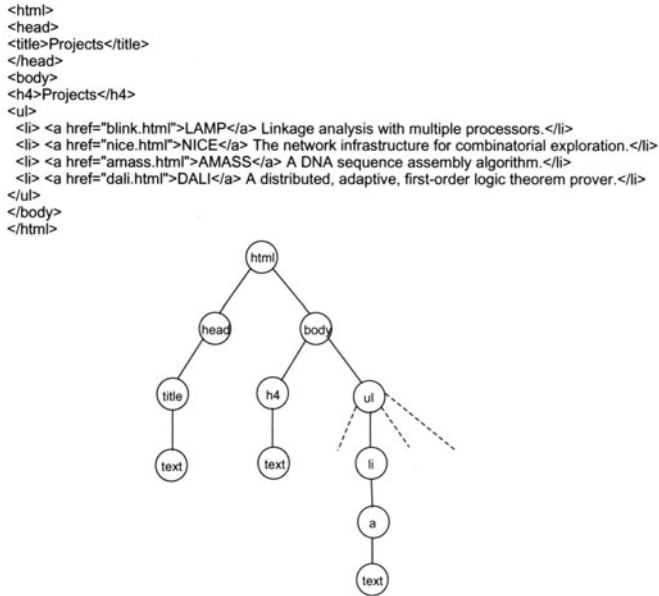


Fig. 2. An HTML page and the corresponding tag tree

## 2.6 Multithreaded Crawlers

A sequential crawling loop spends a large amount of time in which either the CPU is idle (during network/disk access) or the network interface is idle (during CPU operations). Multithreading, where each thread follows a crawling loop, can provide reasonable speed-up and efficient use of available bandwidth. Figure 3 shows a multithreaded version of the basic crawler in Fig. 1. Note that each thread starts by locking the frontier to pick the next URL to crawl. After picking a URL it unlocks the frontier allowing other threads to access it. The frontier is again locked when new URLs are added to it. The locking steps are necessary in order to synchronize the use of the frontier that is now shared among many crawling loops (threads). The model of multithreaded crawler in Fig. 3 follows a standard parallel computing model [18]. Note that a typical crawler would also maintain a shared history data structure for a fast lookup of URLs that have been crawled. Hence, in addition to the frontier it would also need to synchronize access to the history.

The multithreaded crawler model needs to deal with an empty frontier just like a sequential crawler. However, the issue is less simple now. If a thread finds the

frontier empty, it does not automatically mean that the crawler as a whole has reached a dead end. It is possible that other threads are fetching pages and may add new URLs in the near future. One way to deal with the situation is by sending a thread to a sleep state when it sees an empty frontier. When the thread wakes up, it checks again for URLs. A global monitor keeps track of the number of threads currently sleeping. Only when all the threads are in the sleep state does the crawling process stop. More optimizations can be performed on the multithreaded model described here, as for instance to decrease contentions between the threads and to streamline network access.

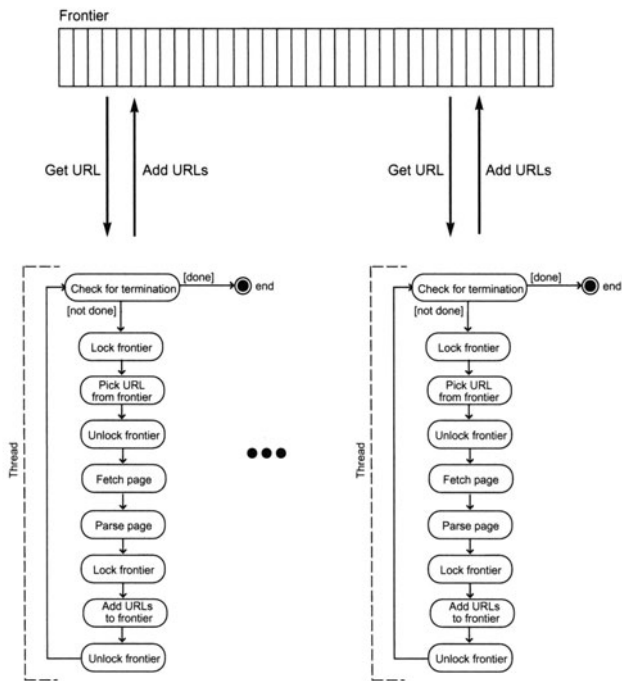


Fig. 3. A multithreaded crawler model

This section described the general components of a crawler. The common infrastructure supports at one extreme a very simple breadth-first crawler and at the other end crawler algorithms that may involve very complex URL selection mechanisms. Factors such as frontier size, page parsing strategy, crawler history, and page repository have been identified as interesting and important dimensions to crawler definitions.

### 3 Crawling Algorithms

We now discuss a number of crawling algorithms that are suggested in the literature. Note that many of these algorithms are variations of the best-first scheme. The difference is in the heuristics they use to score the unvisited URLs, with some algorithms adapting and tuning their parameters before or during the crawl.

#### 3.1 Naive Best-First Crawler

A naive best-first was one of the crawlers detailed and evaluated by the authors in an extensive study of crawler evaluation [22]. This crawler represents a fetched Web page as a vector of words weighted by occurrence frequency. The crawler then computes the cosine similarity of the page to the query or description provided by the user, and scores the unvisited URLs on the page by this similarity value. The URLs are then added to a frontier that is maintained as a priority queue based on these scores. In the next iteration each crawler thread picks the best URL in the frontier to crawl, and returns with new unvisited URLs that are again inserted in the priority queue after being scored based on the cosine similarity of the parent page. The cosine similarity between the page  $p$  and a query  $q$  is computed by:

$$\text{sim}(q, p) = \frac{\mathbf{v}_q \cdot \mathbf{v}_p}{\|\mathbf{v}_q\| \cdot \|\mathbf{v}_p\|}, \quad (1)$$

where  $\mathbf{v}_q$  and  $\mathbf{v}_p$  are term frequency (TF) based vector representations of the query and the page, respectively;  $\mathbf{v}_q \cdot \mathbf{v}_p$  is the dot (inner) product of the two vectors; and  $\|\mathbf{v}\|$  is the Euclidean norm of the vector  $\mathbf{v}$ . More sophisticated vector representation of pages, such as the TF-IDF [32] weighting scheme often used in information retrieval, are problematic in crawling applications because there is no a priori knowledge of the distribution of terms across crawled pages. In a multiple thread implementation the crawler acts like a best- $N$ -first crawler where  $N$  is a function of the number of simultaneously running threads. Thus best- $N$ -first is a generalized version of the best-first crawler that picks  $N$  best URLs to crawl at a time. In our research we have found the best- $N$ -first crawler (with  $N = 256$ ) to be a strong competitor [28, 23], showing clear superiority on the retrieval of relevant pages. Note that the best-first crawler keeps the frontier size within its upper bound by retaining only the best URLs based on the assigned similarity scores.

#### 3.2 SharkSearch

SharkSearch [15] is a version of FishSearch [12] with some improvements. It uses a similarity measure like the one used in the naive best-first crawler for estimating the relevance of an unvisited URL. However, SharkSearch has a more refined notion of potential scores for the links in the crawl frontier. The anchor text, text surrounding the links or *link-context*, and inherited score from ancestors influence the potential scores of links. The ancestors of a URL are the pages that appeared on the crawl path

to the URL. SharkSearch, like its predecessor FishSearch, maintains a depth bound. That is, if the crawler finds unimportant pages on a crawl path it stops crawling farther along that path. To be able to track all the information, each URL in the frontier is associated with a depth and a potential score. The depth bound  $d$  is provided by the user, while the potential score of an unvisited URL is computed as:

$$score(url) = \gamma \cdot inherited(url) + (1 - \gamma) \cdot neighborhood(url), \quad (2)$$

where  $\gamma < 1$  is a parameter, the *neighborhood* score signifies the contextual evidence found on the page that contains the hyperlink URL, and the *inherited* score is obtained from the scores of the ancestors of the URL. More precisely, the *inherited* score is computed as:

$$inherited(url) = \begin{cases} \delta \cdot sim(q, p); & \text{if } sim(q, p) > 0; \\ \delta \cdot inherited(p); & \text{otherwise;} \end{cases} \quad (3)$$

where  $\delta < 1$  is again a parameter,  $q$  is the query, and  $p$  is the page from which the URL was extracted.

The *neighborhood* score uses the anchor text and the text in the “vicinity” of the anchor in an attempt to refine the overall *score* of the URL by allowing for differentiation between links found within the same page. For that purpose, the SharkSearch crawler assigns an *anchor* score and a *context* score to each URL. The *anchor* score is simply the similarity of the anchor text of the hyperlink containing the URL to the query  $q$ , i.e.,  $sim(q, anchor\_text)$ . The *context* score, on the other hand, broadens the context of the link to include some nearby words. The resulting augmented context *aug\_context* is used for computing the *context* score as follows:

$$context(url) = \begin{cases} 1; & \text{if } anchor(url) > 0; \\ sim(q, aug\_context); & \text{otherwise.} \end{cases} \quad (4)$$

Finally, we derive the *neighborhood* score from the *anchor* score and the *context* score as:

$$neighborhood(url) = \beta \cdot anchor(url) + (1 - \beta) \cdot context(url), \quad (5)$$

where  $\beta < 1$  is another parameter. We note that the implementation of SharkSearch would need to preset four different parameters  $d$ ,  $\gamma$ ,  $\delta$  and  $\beta$ . Some values for the same are suggested by [15].

### 3.3 Focused Crawler

A focused crawler based on a hypertext classifier was developed by Chakrabarti et al. [9, 6]. The basic idea of the crawler was to classify crawled pages with categories in a topic taxonomy. To begin, the crawler requires a topic taxonomy such as Yahoo or the Open Directory Project (ODP).<sup>9</sup> In addition, the user provides example URLs

<sup>9</sup> <http://dmoz.org>

of interest (such as those in a bookmark file). The example URLs get automatically classified onto various categories of the taxonomy. Through an interactive process, the user can correct the automatic classification, add new categories to the taxonomy, and mark some of the categories as “good” (i.e., of interest to the user). The crawler uses the example URLs to build a Bayesian classifier that can find the probability ( $\Pr(c|p)$ ) that a crawled page  $p$  belongs to a category  $c$  in the taxonomy. Note that by definition  $\Pr(r|p) = 1$ , where  $r$  is the root category of the taxonomy. A relevance score associated with each crawled page is computed as:

$$R(p) = \sum_{c \in \text{good}} \Pr(c|p). \quad (6)$$

When the crawler is in a “soft” focused mode, it uses the relevance score of the crawled page to score the unvisited URLs extracted from it. The scored URLs are then added to the frontier. Then in a manner similar to the naive best-first crawler, it picks the best URL to crawl next. In the “hard” focused mode, for a crawled page  $p$ , the classifier first finds the leaf node  $c^*$  (in the taxonomy) with maximum probability of including  $p$ . If any of the parents (in the taxonomy) of  $c^*$  are marked as ‘good’ by the user, then the URLs from the crawled page  $p$  are extracted and added to the frontier.

Another interesting element of the focused crawler is the use of a distiller. The distiller applies a modified version of Kleinberg’s algorithm [17] to find topical *hubs*. The hubs provide links to many authoritative sources on the topic. The distiller is activated at various times during the crawl and some of the top hubs are added to the frontier.

### 3.4 Context Focused Crawler

Context focused crawlers [13] use Bayesian classifiers to guide their crawl. However, unlike the focused crawler described above, these classifiers are trained to estimate the link distance between a crawled page and the relevant pages. We can appreciate the value of such an estimation from our own browsing experiences. If we are looking for papers on “numerical analysis,” we may first go to the home pages of math or computer science departments and then move to faculty pages, which may then lead to the relevant papers. A math department Web site may not have the words “numerical analysis” on its home page. A crawler such as the naive best-first crawler would put such a page on low priority and might never visit it. However, if the crawler could estimate that a relevant paper on “numerical analysis” is probably two links away, we would have a way of giving the home page of the math department higher priority than the home page of a law school.

The context focused crawler is trained using a *context graph* of  $L$  layers corresponding to each seed page. The seed page forms layer 0 of the graph. The pages corresponding to the in-links to the seed page are in layer 1. The in-links to the layer 1 pages make up the layer 2 pages, and so on. We can obtain the in-links to pages of any layer by using a search engine. Figure 4 depicts a context graph for

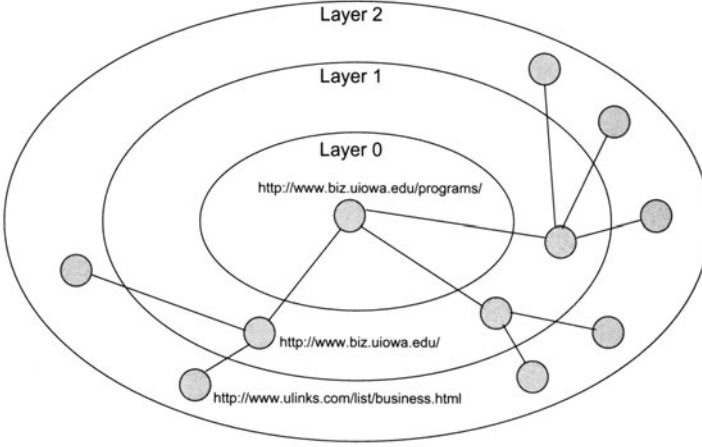
<http://www.biz.uiowa.edu/programs/> as seed. Once the context graphs for all of the seeds are obtained, the pages from the same layer (number) from each graph are combined into a single layer. This gives a new set of layers of what is called a *merged context graph*. This is followed by a feature selection stage where the seed pages (or possibly even layer 1 pages) are concatenated into a single large document. Using the TF-IDF [32] scoring scheme, the top few terms are identified from this document to represent the vocabulary (feature space) that will be used for classification.

A set of naive Bayes classifiers are built, one for each layer in the merged context graph. All the pages in a layer are used to compute  $\Pr(t|c_l)$ , the probability of occurrence of a term  $t$  given the class  $c_l$  corresponding to layer  $l$ . A prior probability,  $\Pr(c_l) = 1/L$ , is assigned to each class where  $L$  is the number of layers. The probability of a given page  $p$  belonging to a class  $c_l$  can then be computed as  $\Pr(c_l|p)$ . Such probabilities are computed for each class. The class with highest probability is treated as the winning class (layer). However, if the probability for the winning class is still less than a threshold, the crawled page is classified into the “other” class. This “other” class represents pages that do not have a good fit with any of the classes of the context graph. If the probability of the winning class does exceed the threshold, the page is classified into the winning class.

The set of classifiers corresponding to the context graph provides us with a mechanism to estimate the link distance of a crawled page from relevant pages. If the mechanism works, the math department home page will get classified into layer 2, while the law school home page will get classified to “others.” The crawler maintains a queue for each class, containing the pages that are crawled and classified into that class. Each queue is sorted by the the probability scores  $\Pr(c_l|p)$ . When the crawler needs a URL to crawl, it picks the top page in the nonempty queue with smallest  $l$ . So it will tend to pick up pages that seem to be closer to the relevant pages first. The out-links from such pages will get explored before the out-links of pages that seem to be far away from the relevant portions of the Web.

### 3.5 InfoSpiders

In InfoSpiders [21, 23], an adaptive population of agents searches for pages relevant to the topic. Each agent is essentially following the crawling loop (Sect. 2) while using an adaptive query list and a neural net to decide which links to follow. The algorithm provides an exclusive frontier for each agent. In a multithreaded implementation of InfoSpiders (see Sect. 5.1) each agent corresponds to a thread of execution. Hence, each thread has a noncontentious access to its own frontier. Note that any of the algorithms described in this chapter may be implemented similarly (one frontier per thread). In the original algorithm (e.g., [21]) each agent kept its frontier limited to the links on the page that was last fetched by the agent. As a result of this limited memory approach the crawler was limited to following the links on the current page, and it was outperformed by the naive best-first crawler on a number of evaluation criteria [22]. Since then a number of improvements (inspired by naive best-first) to the original algorithm have been designed while retaining its capability to learn



**Fig. 4.** A context graph

link estimates via neural nets and focus its search toward more promising areas by selective reproduction. In fact, the redesigned version of the algorithm has been found to outperform various versions of naive best-first crawlers on specific crawling tasks with crawls that are longer than ten thousand pages [23].

The adaptive representation of each agent consists of a list of keywords (initialized with a query or description) and a neural net used to evaluate new links. Each input unit of the neural net receives a count of the frequency with which the keyword occurs in the vicinity of each link to be traversed, weighted to give more importance to keywords occurring near the link (and maximum in the anchor text). There is a single output unit. The output of the neural net is used as a numerical quality estimate for each link considered as input. These estimates are then combined with estimates based on the cosine similarity (Eq. (1)) between the agent's keyword vector and the page containing the links. A parameter  $\alpha$ ,  $0 \leq \alpha \leq 1$ , regulates the relative importance given to the estimates based on the neural net versus the parent page. Based on the combined score, the agent uses a stochastic selector to pick one of the links in the frontier with probability

$$\Pr(\lambda) = \frac{e^{\beta\sigma(\lambda)}}{\sum_{\lambda' \in \phi} e^{\beta\sigma(\lambda')}}, \quad (7)$$

where  $\lambda$  is a URL in the local frontier  $\phi$  and  $\sigma(\lambda)$  is its combined score. The  $\beta$  parameter regulates the greediness of the link selector.

After a new page has been fetched, the agent receives “energy” in proportion to the similarity between its keyword vector and the new page. The agent's neural net can be trained to improve the link estimates by predicting the similarity of the new page, given the inputs from the page that contained the link leading to it. A back-propagation algorithm is used for learning. Such a learning technique provides InfoSpiders with the unique capability to adapt the link-following behaviour in the course of a crawl

by associating relevance estimates with particular patterns of keyword frequencies around links.

An agent's energy level is used to determine whether or not an agent should reproduce after visiting a page. An agent reproduces when the energy level passes a constant threshold. The reproduction is meant to bias the search toward areas (agents) that lead to good pages. At reproduction, the offspring (new agent or thread) receives half of the parent's link frontier. The offspring's keyword vector is also mutated (expanded) by adding the term that is most frequent in the parent's current document. This term addition strategy in a limited way is comparable to the use of classifiers in Sect. 3.4, since both try to identify lexical cues that appear on pages leading up to the relevant pages.

In this section we have presented a variety of crawling algorithms, most of which are variations of the best-first scheme. The readers may pursue Menczer et. al. [23] for further details on the algorithmic issues related with some of the crawlers.

## 4 Evaluation of Crawlers

In a general sense, a crawler (especially a topical crawler) may be evaluated on its ability to retrieve "good" pages. However, a major hurdle is the problem of recognizing these good pages. In an operational environment real users may judge the relevance of pages as these are crawled, allowing us to determine if the crawl was successful or not. Unfortunately, meaningful experiments involving real users for assessing Web crawls are extremely problematic. For instance, the very scale of the Web suggests that in order to obtain a reasonable notion of crawl effectiveness one must conduct a large number of crawls, i.e., involve a large number of users.

Second, crawls against the live Web pose serious time constraints. Therefore crawls other than short-lived ones will seem overly burdensome to the user. We may choose to avoid these time loads by showing the user the results of the full crawl but this again limits the extent of the crawl.

In the not-so-distant future, the majority of the direct consumers of information is more likely to be Web agents working on behalf of humans and other Web agents than humans themselves. Thus it is quite reasonable to explore crawlers in a context where the parameters of crawl time and crawl distance may be beyond the limits of human acceptance imposed by user-based experimentation.

In general, it is important to compare topical crawlers over a large number of topics and tasks. This will allow us to ascertain the statistical significance of particular benefits that we may observe across crawlers. Crawler evaluation research requires an appropriate set of metrics. Recent research reveals several innovative performance measures. But first we observe that there are two basic dimensions in the assessment process, we need a measure of the crawled page's importance, and second we need a method to summarize performance across a set of crawled pages.



## 4.1 Page Importance

Let us enumerate some of the methods that have been used to measure page importance.

1. *Keywords in document*: A page is considered relevant if it contains some or all of the keywords in the query. Also, the frequency with which the keywords appear on the page may be considered [10].
2. *Similarity to a query*: Often a user specifies an information need as a short query. In some cases a longer description of the need may be available. Similarity between the short or long description and each crawled page may be used to judge the page's relevance [15, 22].
3. *Similarity to seed pages*: The pages corresponding to the seed URLs are used to measure the relevance of each page that is crawled [2]. The seed pages are combined together into a single document and the cosine similarity of this document and a crawled page is used as the page's relevance score.
4. *Classifier score*: A classifier may be trained to identify the pages that are relevant to the information need or task. The training is done using the seed (or prespecified relevant) pages as positive examples. The trained classifier will then provide Boolean or continuous relevance scores to each of the crawled pages [9, 13].
5. *Retrieval system rank*:  $N$  different crawlers are started from the same seeds and allowed to run until each crawler gathers  $P$  pages. All of the  $N \cdot P$  pages collected from the crawlers are ranked against the initiating query or description using a retrieval system such as SMART. The rank provided by the retrieval system for a page is used as its relevance score [22].
6. *Link-based popularity*: One may use algorithms, such as PageRank [5] or Hyperlink-Induced Topic Search (HITS) [17], that provide popularity estimates of each of the crawled pages. A simpler method would be to use just the number of in-links to the crawled page to derive similar information [10, 2]. Many variations of link-based methods using topical weights are choices for measuring topical popularity of pages [4, 7].

## 4.2 Summary Analysis

Given a particular measure of page importance we can summarize the performance of the crawler with metrics that are analogous to the information retrieval (IR) measures of *precision* and *recall*. Precision is the fraction of retrieved (crawled) pages that are relevant, while recall is the fraction of relevant pages that are retrieved (crawled). In a usual IR task the notion of a relevant set for recall is restricted to a given collection or database. Considering the Web to be one large collection, the relevant set is generally unknown for most Web IR tasks. Hence, explicit recall is hard to measure. Many authors provide precision-like measures that are easier to compute in order to evaluate the crawlers. We will discuss a few such precision-like measures:

1. *Acquisition rate*: In cases where we have Boolean relevance scores we could measure the explicit rate at which "good" pages are found. Therefore, if 50

relevant pages are found in the first 500 pages crawled, then we have an acquisition rate or *harvest rate* [1] of 10% at 500 pages.

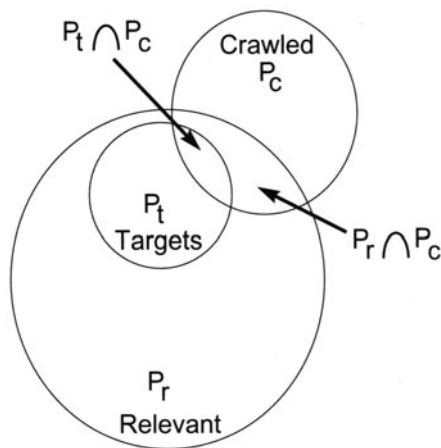
2. *Average relevance*: If the relevance scores are continuous they can be averaged over the crawled pages. This is a more general form of harvest rate [9, 22, 8]. The scores may be provided through simple cosine similarity or a trained classifier. Such averages (Fig. 6(a)) may be computed over the progress of the crawl (first 100 pages, first 200 pages, and so on) [22]. Sometimes running averages are calculated over a window of a few pages (e.g., the last 50 pages from a current crawl point) [9].

Since measures analogous to recall are hard to compute for the Web, authors resort to indirect indicators for estimating recall. Some such indicators are:

1. *Target recall*: A set of known relevant URLs is split into two disjoint sets—*targets* and *seeds*. The crawler is started from the seeds pages and the recall of the targets is measured. The target recall is computed as

$$target\_recall = \frac{|\mathcal{P}_t \cap \mathcal{P}_c|}{|\mathcal{P}_t|}$$

, where  $\mathcal{P}_t$  is the set of target pages, and  $\mathcal{P}_c$  is the set of crawled pages. The recall of the target set is used as an estimate of the recall of relevant pages. Figure 5 gives a schematic justification of the measure. Note that the underlying assumption is that the targets are a random subset of the relevant pages.



**Fig. 5.** The performance metric  $|\mathcal{P}_t \cap \mathcal{P}_c| / |\mathcal{P}_t|$  as an estimate of  $|\mathcal{P}_r \cap \mathcal{P}_c| / |\mathcal{P}_r|$

2. *Robustness*: The seed URLs are split into two disjoint sets  $S_a$  and  $S_b$ . Each set is used to initialize an instance of the same crawler. The overlap in the pages crawled starting from the two disjoint sets is measured. A large overlap is interpreted as *robustness* of the crawler in covering relevant portions of the Web [9, 6].

There are other metrics that measure the crawler performance in a manner that combines both precision and recall. For example, *search length* [21] measures the number of pages crawled before a certain percentage of the relevant pages are retrieved.

Figure 6 shows an example of performance plots for two different crawlers. The crawler performance is depicted as a trajectory over time (approximated by crawled pages). The naive best-first crawler is found to outperform the breadth-first crawler based on evaluations over 159 topics with 10,000 pages crawled by each crawler on each topic (hence the evaluation involves millions of pages).

In this section we have outlined methods for assessing page importance and measures to summarize crawler performance. When conducting a fresh crawl experiment it is important to select an evaluation approach that provides a reasonably complete and sufficiently detailed picture of the crawlers being compared.

## 5 Applications

We now briefly review a few applications that use crawlers. Our intent is not to be comprehensive but instead to simply highlight their utility.

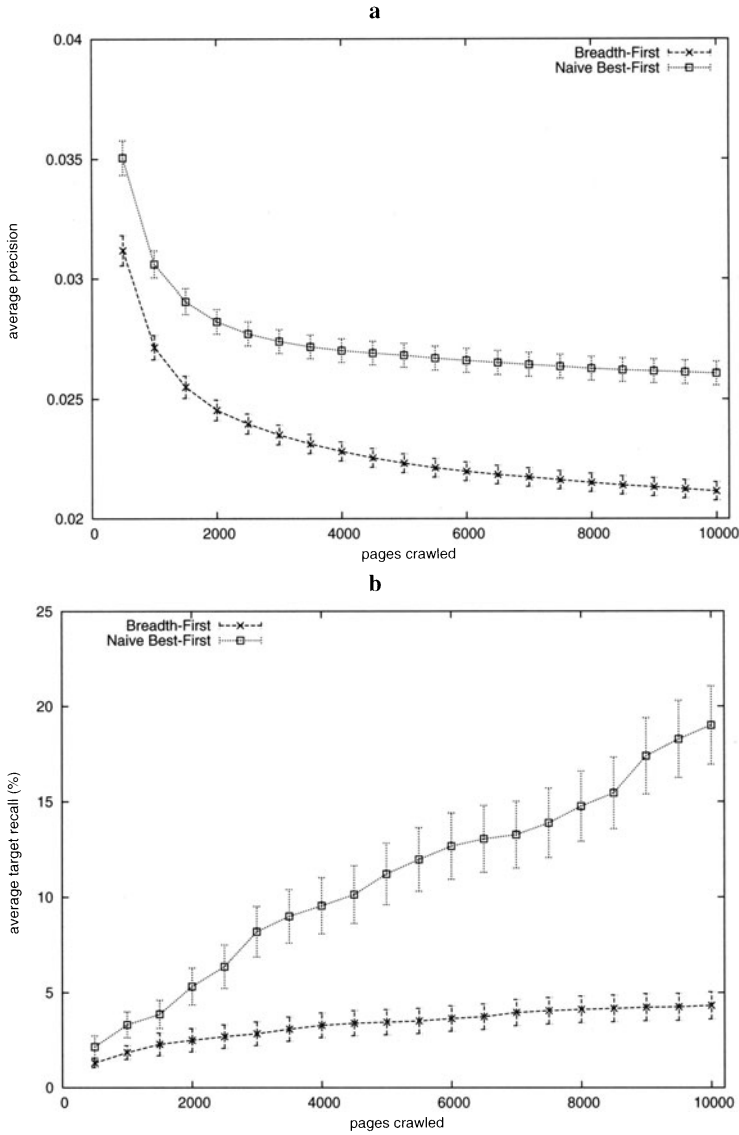
### 5.1 MySpiders: Query-Time Crawlers

MySpiders [26] is a Java applet that implements the InfoSpiders and the naive best-first algorithms. Multithreaded crawlers are started when a user submits a query. Results are displayed dynamically as the crawler finds “good” pages. The user may browse the results while the crawling continues in the background. The multithreaded implementation of the applet deviates from the general model specified in Fig. 3. In line with the autonomous multiagent nature of the InfoSpiders algorithm (Sect.3.5), each thread has a separate frontier. This applies to the naive best-first algorithm as well. Hence, each thread is more independent with noncontentious access to its frontier. The applet allows the user to specify the crawling algorithm and the maximum number of pages to fetch. In order to initiate the crawl, the system uses the Google Web API<sup>10</sup> to obtain a few seed pages. The crawler threads are started from each of the seeds, and the crawling continues until the required number of pages are fetched or the frontier is empty. Figure 7 shows MySpiders working on a user query using the InfoSpiders algorithm.

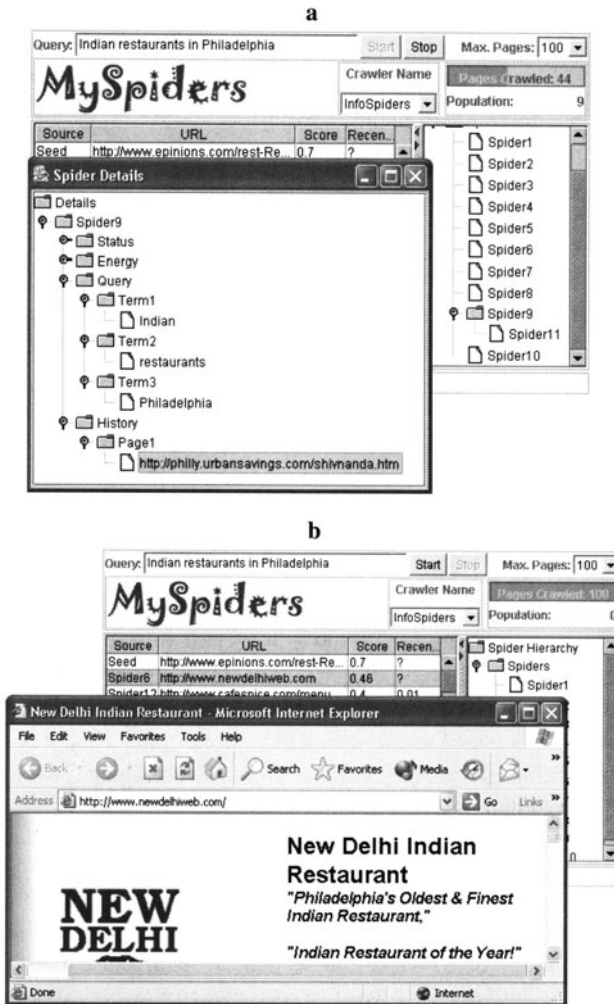
### 5.2 CORA: Building Topic-Specific Portals

A topical crawler may be used to build topic-specific portals such as sites that index research papers. One such application developed by McCallum et al. [20] collected and maintained research papers in Computer Science (CORA). The crawler used by the application is based on reinforcement learning (RL) that allows for finding

<sup>10</sup> <http://www.google.com/apis>



**Fig. 6.** Performance Plots: **(a)** average precision (similarity to topic description); **(b)** average target recall. The averages are calculated over 159 topics and the error bars show  $\pm 1$  standard error. One-tailed  $t$ -test for the alternative hypothesis that the naive best-first crawler outperforms the breadth-first crawler (at 10,000 pages) generates  $p$  values that are  $< 0.01$  for both performance metrics



**Fig. 7.** The user interface of *MySpiders* during a crawl using the InfoSpiders algorithm. (a) In the search process, Spider 9 has reproduced and its progeny is visible in the expandable tree (right). A spider's details are revealed by clicking on it on the tree (left). (b) At the end of the crawl, one of the top hits is found by a spider (and it is not one of the seeds). The hit is viewed by clicking its URL in the results frame

crawling policies that lead to immediate as well as long-term benefits. The benefits are discounted based on how far away they are from the current page. Hence, a hyperlink that is expected to immediately lead to a relevant page is preferred over one that is likely to bear fruit after a few links. The need to consider future benefit along a crawl path is motivated by the fact that lexical similarity between pages falls rapidly with increasing link distance. Therefore, as noted earlier, a math department home page that leads to a numerical analysis paper may provide very little lexical signal

to a naive best-first crawler that is searching for the paper. Hence, the motivation of the RL crawling algorithm is similar to that of the context focused crawler. The RL crawler was trained using known paths to relevant pages. The trained crawler is then used to estimate the benefit of following a hyperlink.

### 5.3 Mapuccino: Building Topical Site Maps

One approach to building site maps is to start from a seed URL and crawl in a breadth-first manner until a certain number of pages have been retrieved or a certain depth has been reached. The site map may then be displayed as a graph of connected pages. However, if we are interested in building a site map that focuses on a certain topic, then the above-mentioned approach will lead to a large number of unrelated pages as we crawl to greater depths or fetch more pages. Mapuccino [15] corrects this by using SharkSearch (Sect. 3.2) to guide the crawler and then build a visual graph that highlights the relevant pages.

### 5.4 Letizia: a Browsing Agent

Letizia [19] is an agent that assists a user during browsing. While the user surfs the Web, Letizia tries to understand user interests based on the pages being browsed. The agent then follows the hyperlinks starting from the current page being browsed to find pages that could be of interest to the user. The hyperlinks are crawled automatically and in a breadth-first manner. The user is not interrupted, but pages of possible interest are suggested only when she needs recommendations. The agent makes use of topical locality on the Web [11] to provide context-sensitive results.

### 5.5 Other Applications

Crawling in general and topical crawling in particular is being applied for various other applications, many of which do not appear as technical papers. For example, business intelligence has much to gain from topical crawling. A large number of companies have Web sites where they often describe their current objectives, future plans, and product lines. In some areas of business, there are a large number of start-up companies that have rapidly changing Web sites. All these factors make it important for various business entities to use sources other than the general-purpose search engines to keep track of relevant and publicly available information about their potential competitors or collaborators [27].

Crawlers have also been used for biomedical applications like finding relevant literature on a gene [33]. On a different note, there are some controversial applications of crawlers such as extracting e-mail addresses from Web sites for spamming.

## 6 Conclusion

Because of the dynamism of the Web, crawling forms the backbone of applications that facilitate Web information retrieval. While the typical use of crawlers has been for

creating and maintaining indexes for general-purpose search engines, diverse usage of topical crawlers is emerging both for client and server-based applications. Topical crawlers are becoming important tools to support applications such as specialized Web portals, online searching, and competitive intelligence. A number of topical crawling algorithms have been proposed in the literature. Often the evaluation of these crawlers is done by comparing a few crawlers on a limited number of queries/tasks without considerations of statistical significance. Anecdotal results, while important, do not suffice for thorough performance comparisons. As the Web crawling field matures, the disparate crawling strategies will have to be evaluated and compared on common tasks through well-defined performance measures.

In the future, we see more sophisticated usage of hypertext structure and link analysis by the crawlers. For a current example, Chakrabarti et. al. [8] have suggested the use of the pages' HTML tag tree or DOM structure for focusing a crawler. While they have shown some benefit of using the DOM structure, a thorough study on the merits of using the structure (in different ways) for crawling is warranted [24]. Topical crawlers depend on various cues from crawled pages to prioritize the fetching of unvisited URLs. A good understanding of the relative importance of cues such as the link context, linkage (graph) structure, ancestor pages, and so on is also needed [16]. Another potential area of research is stronger collaboration between search engines and crawlers [25], and among the crawlers themselves. The scalability benefits of distributed topical crawling [9, 21] are yet to be fully realized. Can crawlers help a search engine to focus on user interests? Can a search engine help a crawler to focus on a topic? Can a crawler on one machine help a crawler on another? Many such questions will motivate future research and crawler applications.

## Acknowledgments

The authors would like thank the anonymous referees for their valuable suggestions. This work is funded in part by NSF CAREER Grant No. IIS-0133124 to FM.

## References

1. C. C. Aggarwal, F. Al-Garawi, and P. S. Yu. Intelligent crawling on the World Wide Web with arbitrary predicates. In *WWW10*, Hong Kong, May 2001.
2. B. Amento, L. Terveen, and W. Hill. Does "authority" mean quality? Predicting expert quality ratings of web documents. In *Proc. 23th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, Athens, Greece, 2000.
3. A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the Web. *ACM Transactions on Internet Technology*, 1(1), 2001.
4. K. Bharat and M.R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, 1998.
5. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30:107–117, 1998.

6. S. Chakrabarti. *Mining the Web*. Morgan Kaufmann, 2003.
7. S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource list compilation by analyzing hyperlink structure and associated text. In *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, 1998.
8. S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *WWW2002*, Hawaii, May 2002.
9. S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific Web resource discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.
10. J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks*, 30:161–172, 1998.
11. B.D. Davison. Topical locality in the web. In *Proc. 23rd Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, Athens, Greece, 2000.
12. P. M. E. De Bra and R. D. J. Post. Information retrieval in the World Wide Web: Making client-based searching feasible. In *Proc. 1st International World Wide Web Conference*, 1994.
13. M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *Proc. 26th International Conference on Very Large Databases (VLDB 2000)*, pages 527–534, Cairo, Egypt, 2000.
14. D. Eichmann. Ethical Web agents. In *Second International World-Wide Web Conference*, pages 3–13, Chicago, Illinois, 1994.
15. M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhim, and S. Ur. The shark-search algorithm — An application: Tailored Web site mapping. In *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, 1998.
16. J. Johnson, K. Tsoutsoulouklis, and C.L. Giles. Evolving strategies for focused web crawling. In *Proc. 12th Intl. Conf. on Machine Learning (ICML-2003)*, Washington DC, 2003.
17. J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
18. V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings, 1994.
19. H. Lieberman, F. Christopher, and L. Weitzman. Exploring the Web with reconnaissance agents. *Communications of the ACM*, 44:69–75, August 2001.
20. A.K. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.
21. F. Menczer and R. K. Belew. Adaptive retrieval agents: Internalizing local context and scaling up to the Web. *Machine Learning*, 39(2–3):203–242, 2000.
22. F. Menczer, G. Pant, M. Ruiz, and P. Srinivasan. Evaluating topic-driven Web crawlers. In *Proc. 24th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, New Orleans, Louisiana, 2001.
23. F. Menczer, G. Pant, and P. Srinivasan. Topical web crawlers: Evaluating adaptive algorithms. *To appear in ACM Trans. on Internet Technologies*, 2003. <http://dollar.biz.uiowa.edu/~fil/Papers/TOIT.pdf>.
24. G. Pant. Deriving link-context from HTML tag tree. In *8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2003.
25. G. Pant, S. Bradshaw, and F. Menczer. Search engine-crawler symbiosis: Adapting to community interests. In *Proc. 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2003)*, Trondheim, Norway, 2003.
26. G. Pant and F. Menczer. MySpiders: Evolve your own intelligent Web crawlers. *Autonomous Agents and Multi-Agent Systems*, 5(2):221–229, 2002.



27. G. Pant and F. Menczer. Topical crawling for business intelligence. In *Proc. 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2003)*, Trondheim, Norway, 2003.
28. G. Pant, P. Srinivasan, and F. Menczer. Exploration versus exploitation in topic driven crawlers. In *WWW02 Workshop on Web Dynamics*, Honolulu, Hawaii, 2002.
29. M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
30. S. RaviKumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the Web graph. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 57–65, Redondo Beach, CA, Nov. 2000.
31. J. Rennie and A. K. McCallum. Using reinforcement learning to spider the Web efficiently. In *Proc. 16th International Conf. on Machine Learning*, pages 335–343, Bled, Slovenia, 1999.
32. G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY, 1983.
33. P. Srinivasan, J. Mitchell, O. Bodenreider, G. Pant, and F. Menczer. Web crawling agents for retrieving biomedical information. In *NETTAB: Agents in Bioinformatics*, Bologna, Italy, 2002.

---

# Combining Link and Content Information in Web Search

Matthew Richardson and Pedro Domingos

Department of Computer Science and Engineering  
University of Washington,  
Seattle, WA 98115, USA  
{mattr, pedrod}@cs.washington.edu

**Summary.** As the World Wide Web has grown, search engines have become the preferred method for finding information on the Web; it is now almost impossible to find specific information without them. This has given rise to a problem: How can we automatically determine the quality and relevance of a Web page to a particular query? The original search engines used the content of a page to determine its relevance, but recently it was found that results could be greatly improved by incorporating information gleaned from the link structure as well. In this chapter, we describe two of the most well-known algorithms that do this: *HITS* [17] and *PageRank* [18], and also survey some of their improvements. We then introduce our algorithm, *Query-Dependent PageRank*, which maintains query-time efficiency while alleviating the problem of topic drift. Experiments on two large subsets of the Web indicate that our algorithm significantly outperforms PageRank in the (human-rated) quality of the pages returned, while remaining efficient enough to be used in today's large search engines. After presenting these results and a discussion of scalability, we leave the reader with some open questions and possible directions for future work.

## 1 Introduction

Traditional information retrieval measures, such as Term Frequency Inverse Document Frequency (TFIDF) [20], rate a document highly if the query terms occur frequently in it. These can give poor results on the Web, with its vast scale and highly variable content quality. Recently, however, it was found that search results can be greatly improved by utilizing the information contained in the link structure between pages.

The two best-known algorithms that do this are Hyperlink-Induced Topic Search (HITS) [17] and PageRank [18]. The latter is used in the highly successful Google search engine [3]. The heuristic underlying both of these approaches is that pages with many in-links are likely to be of higher quality than pages with few in-links. This is based on the assumption that the author of a page will only include links to pages that he or she believes are of high quality. But both HITS and PageRank have downsides. HITS is too slow at query time to be used in practice in a large-scale search engine.

PageRank (and, to a lesser extent, HITS) suffers from *topic drift*, whereby it may return pages that, although they are of high quality, are only peripherally related to the query. This occurs because only the link structure between pages is considered by the computation – the content of pages is ignored.

We first discuss the information contained in the link structure of the Web. We review HITS and some of its recent improvements. We then present the PageRank algorithm and show how it is prone to topic drift. In the balance of the chapter we present our query-dependent version of PageRank, which tackles the problem of topic drift with clean probabilistic semantics. Further, by incorporating query relevance directly into PageRank, we eliminate the ad hoc posterior step of merging a page's PageRank and query relevance scores. Our results show that a query-dependent PageRank can be implemented efficiently and returns better search results than the traditional (query-independent) PageRank. We conclude with some open questions and ideas for future research.

## 2 Links Contain Information

An important characteristic that differentiates Web pages from simple documents is that they are interconnected. Initial attempts at Web search considered each page to be independent, but in fact the hyperlinks between pages contain useful information on the topical content and quality of pages.

### 2.1 Topical Content

A Web page typically links to other pages on similar or related topics [11]. Hence, when inferring the topic of a page, it is useful to consider not only the contents of the page itself, but also the contents of the pages it links to and those that link to it (a page's *neighbors*). One simple approach is to include the contents of the neighboring pages when computing the topic of a page. Chakrabarti et al. [8] found that this actually *decreased* classification accuracy, which they attributed to the fact that links are noisy – a page tends to link to pages on a similar topic, but will also typically contain a few links to pages on completely different topics (“Free Speech Online” or “Yahoo”, for example). Further, this does not take advantage of the (albeit weaker) correlation between the topic of a page and its neighbors' neighbors, or their neighbors, and so on. Notice that this is a circular problem: the topic of a page depends on its neighbors, but their topics depend on it. Chakrabarti et al. thus use a Markov random field in which each node represents a page, the state of a node is the page's topic, and edges connect any two nodes whose pages are neighbors. The best assignment of topics to pages can be solved using Markov random field methods such as relaxation labeling. With this technique, they were able to reduce the error rate on a Web page classification task from about 36% to about 26%. For the remainder of the chapter, we will focus on another piece of information provided by links, that of quality.

## 2.2 Quality

The problem when searching the Web is that it is too vast, and queries are too underspecified. A typical query can easily match thousands or even hundreds of thousands of pages. A good search engine needs to order these results by some measure of quality and relevance to the query. Methods such as TFIDF help provide the relevance of a page to a query but do not provide information on its quality.

In general, links are created by people. As such, they are indicative of the quality of the pages to which they point – when creating a page, an author presumably chooses to link to pages which s/he deems to be of good quality. By taking advantage of this information we can estimate the quality of each page based on the link structure around it. In the following two sections we introduce HITS and PageRank, two algorithms for doing this. Further introduction to these methods, as well as common techniques for modeling and mining textual content, can be found in [4] and [5].

## 3 HITS

HITS [17] is based on the notion that pages can be primarily categorized into two types: *hubs* (pages that point to many pages of high quality) and *authorities* (pages of high quality). Given a query, HITS first invokes a traditional search engine to obtain a set of pages relevant to it (the *root set*). This set is then expanded to include all pages which link to or are linked to by one of its pages.<sup>1</sup>

Each page  $x$  is assigned a hub (or index) score  $h(x)$  and authority score  $a(x)$  (initially set to 1) which are updated according to the equations:

$$a(i) = \sum_{j \in \mathbf{B}_i} h(j) \quad h(i) = \sum_{j \in \mathbf{F}_i} a(j), \quad (1)$$

where  $\mathbf{F}_i$  is the set of pages page  $i$  links to, and  $\mathbf{B}_i$  is the set of pages that link to page  $i$ . The equations are iterated until they converge. Notice that, from the equations, hubs and authorities are defined recursively; a hub is a page that points to many authorities, and an authority is a page that is pointed to by many hubs.

This computation takes advantage of the fact that (typically) links are human generated, and the choice of pages to link to reflects an author's opinion of their quality. It also uses the fact that many pages on the WWW are organized into this hub and authority structure.

### 3.1 Topic Drift in HITS

The HITS computation for a given query only depends on it via the initial root set. The pages used to expand the root set may or may not be related to the query, and the

<sup>1</sup> Typically, the pages that link to a URL are found by querying a search engine. Many search engines, such as Altavista ([www.altavista.com](http://www.altavista.com)), allow queries of the form `link:URL`, which returns all (up to a limit) pages which link to a given URL. The pages that a page links to are typically found by retrieving the page and parsing it.

computation of good hubs and authorities depends only on the link structure among those pages, not their content. As a result, HITS suffers from *topic drift*, a drift away from the query topic because some peripheral topic is more popular and/or has better connected pages.

A variety of research has been conducted on combating this problem. One common technique is to weight each edge according to some function of the query terms and the pages it connects. This is the approach taken by IBM's Clever system [7, 6], in which an edge is given a higher weight if the text near the link contains the query terms. The rationale behind this is that, for a particular topic, a hub is really only conferring authority to pages that it specifically points to in reference to that topic. With weighted edges, Eq. 1 simply becomes:

$$a(i) = \sum_{j \in \mathbf{B}_i} w(j \rightarrow i)h(j), \quad h(i) = \sum_{j \in \mathbf{F}_i} w(i \rightarrow j)a(j), \quad (2)$$

where  $w(i \rightarrow j)$  is the weight assigned to the link from page  $i$  to page  $j$ .

Clever further reduces topic drift by breaking large hub pages into *pagelets*. A hub may cover multiple topics (e.g., a professor's "favorite links" page, which lists pages on machine learning as well as bicycling), or varying specialties of a broad topic (e.g., a hub about international travel, with separate sections devoted to Asia and Europe). Therefore dividing the page into sections allows the hub scores for each section to be more accurate and query specific [6]. Similar work includes that of Bharat and Henzinger [2], who propose heuristic methods for weighting links based on pages' relevance to the root set. Cohn and Hofmann [10] introduced a joint probabilistic model that combined the content of pages (using probabilistic latent semantic analysis [15]) and the connectivity between them (using a probabilistic version of HITS called PHITS [9]). A joint model of content and connectivity significantly reduces the problems of topic drift, and also opens up opportunities for a variety of interesting new mining tasks.

### 3.2 Speed of HITS

HITS has an additional drawback: the hub and authority scores are computed at query time. Though the iterative computation (Eq. 1) typically takes only about one second [7], the retrieval of the relevant pages (the root set) can take up to 30 minutes [2], which would be infeasible for large-scale Web search engines.

As of this writing, Google, a popular search engine, requires approximately 0.1 seconds to answer a given query. Yet even at this speed, the company requires over 10,000 computers to serve the 150 million search requests per day it receives.<sup>2</sup> It would be prohibitively expensive for a popular search engine to require even seconds per query. If used "within" the search engine, HITS may have access to the contents of crawled pages, making it much faster, but still an order of magnitude slower than Google. Further, to be efficient, the set of pages upon which HITS is performed must

<sup>2</sup> <http://www.google.com/press/highlights.html>

be kept small. This limits its efficient use to queries that return a small number of pages and have relatively local link structure.

Bharat et al. propose using a *connectivity server* [1], which crawls and serves page linkage information. Using such a server, HITS may simply retrieve the link structure of its root set, rather than having to retrieve every page, thus making it much more responsive. However, using such a server is incompatible with the techniques outlined in the previous section for preventing topic drift, which require knowing the content of each page (unless the computation is being performed “within” the search engine).

In the next section, we introduce PageRank, another method for exploiting the information contained in the links between pages. In contrast to HITS, PageRank computes a single measure of quality for each page before any query is issued. This measure is then combined with a traditional information retrieval score at query time. This has the advantage of much greater efficiency, but has the disadvantage that there is no clear and principled way to combine PageRank with a measure of query relevance,<sup>3</sup> leaving it open to the same problems of topic drift that were encountered by HITS.

## 4 PageRank: The Random Surfer

Imagine a Web surfer who jumps from Web page to Web page, choosing with uniform probability which link to follow at each step. In order to reduce the effect of dead ends or endless cycles, the surfer will occasionally jump to a random page with some small probability  $1 - \beta$ , or when on a page with no out-links. To reformulate this in graph terms, consider the Web as a directed graph, where nodes represent Web pages, and edges between nodes represent links between Web pages. Let  $\mathbf{W}$  be the set of nodes  $N=|\mathbf{W}|$ ,  $\mathbf{F}_i$  be the set of pages page  $i$  links to, and  $\mathbf{B}_i$  be the set of pages which link to page  $i$ . For pages that have no out-links we add a link to all pages in the graph.<sup>4</sup> In this way, probability mass that is lost due to pages with no out-links is redistributed uniformly to all pages. If averaged over a sufficient number of steps, the probability the surfer is on page  $j$  at some point in time is given by the formula:

$$P(j) = \frac{(1 - \beta)}{N} + \beta \sum_{i \in \mathbf{B}_j} \frac{P(i)}{|\mathbf{F}_i|}. \quad (3)$$

The PageRank score for node  $j$  is defined as probability  $PR(j) = P(j)$ . Because Eq. 3 is recursive, it must be iteratively evaluated until  $P(j)$  converges. Typically, the initial distribution for  $P(j)$  is uniform. PageRank is equivalent to the primary eigenvector of the transition matrix  $\mathbf{Z}$ :

<sup>3</sup> As we will see later in this chapter, the method for combining PageRank with text-based query relevance has a large effect on the results. Exactly how Google does this combination is, so far, an unpublished trade secret.

<sup>4</sup> For each page  $s$  with no out-links, we set  $\mathbf{F}_s = \{\text{all } N \text{ nodes}\}$ , and for all other nodes augment  $\mathbf{B}_i$  with  $s$ ,  $(\mathbf{B}_i \cup \{s\})$ .

$$\mathbf{Z} = (1 - \beta) \left[ \frac{1}{N} \right]_{N \times N} + \beta \mathbf{M}, \quad (4)$$

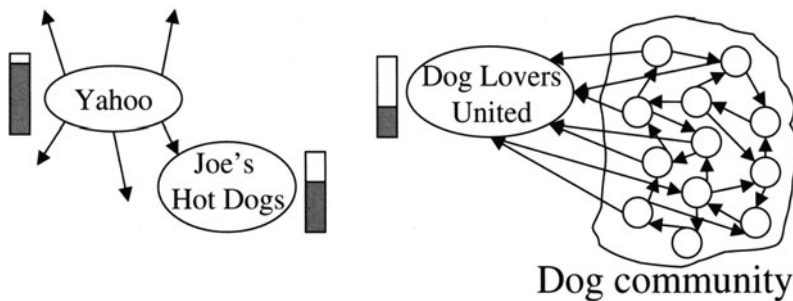
with

$$M_{ji} = \begin{cases} \frac{1}{|\mathbf{F}_i|}, & \text{if there is an edge from } i \text{ to } j, \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

One iteration of Eq. 3 is equivalent to computing  $x^{k+1} = \mathbf{Z}x^k$ , where  $x_j^k = P(j)$  at iteration  $k$ . After convergence, we have  $x^{K+1} = x^K$ , or  $x^K = \mathbf{Z}x^K$ , which means  $x^K$  is an eigenvector of  $\mathbf{Z}$ . Furthermore, since the columns of  $\mathbf{Z}$  are normalized,  $x$  has an eigenvalue of 1.

#### 4.1 Topic Drift in PageRank

PageRank is query independent, which allows it to be computed off-line. At query time, the PageRank may simply be looked up, which allows for very quick query-time response, but it also means that the PageRank of a page is completely independent of its content and the query itself. Content is brought back into the picture by the combination of PageRank with some measure of query relevance. As a result, PageRank suffers from topic drift.



**Fig. 1.** Topic drift. The bars represent the PageRank of each page

Figure 1 demonstrates this problem. A page containing the word “dog”, which is linked to from a page with very high PageRank, such as Yahoo, can end up with a very high PageRank. Instead, we would expect that the best page about dogs would be one that is linked to from many other pages in the “dog community”: a collection of pages about dogs that are interlinked with each other.

Some work has been done to combat topic drift in PageRank. Haveliwala has introduced the *topic-sensitive PageRank* [13], which has some similarities to our own *query-dependent PageRank*, which we describe below. In topic-sensitive PageRank, one PageRank is calculated for each of a number of topics, which are then combined at query time depending on how the query relates to the topics. When calculating

a particular topic-specific PageRank, the random surfer will occasionally jump to one of a set of pages on a particular topic (taken from the Open Directory Project<sup>5</sup>) rather than any page. This has the effect of “grounding” the PageRank to that topic. Recent work by Jeh and Windom [16] improves the scalability of this approach. In the next section, we introduce our method for combating topic drift, *query-dependent PageRank*.

Tables 1 and 2 summarize some of the main advantages, disadvantages, and characteristics of the various link-based Web search methods surveyed in this paper.

**Table 1.** Summary of some methods for using link structure to improve Web search

Algorithm	Pros	Cons
<b>HITS</b> [17]	Seeded by query/content	Computation ignores content Slow at query time
Auto. Resource Compilation [7]	Weigh links by relevance of anchor text to query	Ad hoc
The Missing Link [10]	Consistent prob. semantics to incorporate content	Lack of scalability
Connectivity Server [1]	Retrieve link information quickly	Still too slow for search engine Incompatible with link weighting by content
<b>PageRank</b> [18]	Fast at query time	Combination with content is problematic
TS-PageRank [13]	Incorporates content in part of calculation	Coarse notion of content
QD-PageRank [19]	Sound, fine-grained incorporation of content	More computation at crawl time

**Table 2.** Summary of computations performed by link-based Web search methods

Algorithm	Offline	Query Time
HITS	—	Crawl and computation
HITS + connectivity Server	Crawl	Retrieve graph and compute
PageRank	Computation	Retrieve score
QD-PageRank	Computation	Retrieve score per query term and add

<sup>5</sup> <http://www.dmoz.org>



## 5 Directed Surfer Model

PageRank rates a page highly if it is at the center of a large sub-Web (i.e., if many pages point to it, many other pages point to those, etc.). Intuitively, however, the best pages should be those that are at the center of a large sub-Web *relevant to the query*. This section introduces our algorithm, which formalizes this intuition while, like PageRank, does most of its computations at crawl time [19].

The PageRank of a page can be viewed as the rate at which a surfer would visit that page, if it surfed the Web indefinitely, blindly jumping from page to page. We propose a more intelligent surfer that, more like a human surfer, probabilistically hops from page to page, depending on the content of the pages and the query terms the surfer is looking for. The resulting probability distribution over pages is:

$$P_q(j) = (1 - \beta)P'_q(j) + \beta \sum_{i \in \mathbf{B}_j} P_q(i)P_q(i \rightarrow j), \quad (6)$$

where  $P_q(i \rightarrow j)$  is the probability that the surfer transitions to page  $j$  given that he is on page  $i$  and is searching for the query  $q$ .  $P'_q(j)$  specifies where the surfer chooses to jump when not following links.  $P_q(j)$  is the resulting probability distribution over pages and corresponds to the *query-dependent PageRank* score ( $\text{QD-PageRank}_q(j) \equiv P_q(j)$ ). As with PageRank, QD-PageRank is determined by iterative evaluation of Eq. 6 from some initial distribution, and is equivalent to the primary eigenvector of the transition matrix  $\mathbf{Z}_q$ , where  $Z_{qji} = (1 - \beta)P'_q(j) + \beta P_q(i \rightarrow j)$ . Although  $P_q(i \rightarrow j)$  and  $P'_q(j)$  are arbitrary distributions, we will focus on the case where both probability distributions are derived from  $R_q(j)$ , a measure of relevance of page  $j$  to query  $q$ :

$$P'_q(j) = \frac{R_q(j)}{\sum_{k \in \mathbf{W}} R_q(k)}, \quad (7)$$

$$P_q(i \rightarrow j) = \frac{R_q(j)}{\sum_{k \in \mathbf{F}_i} R_q(k)}. \quad (8)$$

In other words, when choosing among multiple out-links from a page, the directed surfer tends to follow those that lead to pages whose content has been deemed relevant to the query (according to  $R_q$ ). Similarly to PageRank, when a page has no out-links (or the out-links all have zero relevance), we implicitly add links from that page to all other pages in the network. On such a page, the surfer thus chooses a new page to jump to according to the distribution  $P'_q(j)$ .

When given a multiple-term query,  $\mathbf{Q} = \{q_1, q_2, \dots\}$ , the surfer selects a  $q$  according to some probability distribution  $P(q)$  and uses that term to guide its behavior (according to Eq. 6) for a large number of steps.<sup>6</sup> It then selects another term accord-

<sup>6</sup> However, many steps are needed to reach convergence of Eq. 6.

ing to the distribution to determine its behavior, and so on. The resulting distribution over visited Web pages is QD-PageRank<sub>Q</sub> and is given by:

$$\text{QD-PageRank}_Q(j) \equiv P_Q(j) = \sum_{q \in Q} P(q)P_q(j) \quad (9)$$

For standard PageRank, the PageRank vector is equivalent to the primary eigenvector of the matrix  $\mathbf{Z}$ . The vector of single-term QD-PageRank<sub>q</sub> is again equivalent to the primary eigenvector of the matrix  $\mathbf{Z}_q$ . An interesting question that arises is whether the QD-PageRank<sub>Q</sub> vector is equivalent to the primary eigenvector of a matrix (corresponding to the combination performed by Eq. 9). In fact, this is not the case. Instead, the primary eigenvector of  $\mathbf{Z}_Q$  corresponds to the QD-PageRank obtained by a random surfer who, *at each step*, selects a new query according to the distribution  $P(q)$ . However, QD-PageRank<sub>Q</sub> is approximately equal to the PageRank that results from this single-step surfer, for the following reason.

Let  $\mathbf{x}_q$  be the L2-normalized primary eigenvector for matrix  $\mathbf{Z}_q$  (note that element  $j$  of  $\mathbf{x}_q$  is QD-PageRank<sub>q</sub>( $j$ )). Since  $\mathbf{x}_q$  is the primary eigenvector for  $\mathbf{Z}_q$ , we have [12]:  $\forall q, r \in Q : \|\mathbf{Z}_q \mathbf{x}_q\| \geq \|\mathbf{Z}_q \mathbf{x}_r\|$ . Thus, to a first degree of approximation,  $\mathbf{Z}_q \sum_{r \in Q} \mathbf{x}_r \approx \kappa \mathbf{Z}_q \mathbf{x}_q$ . Suppose  $P(q) = 1/|Q|$ . Consider  $\mathbf{x}_Q = \sum_{q \in Q} P(q) \mathbf{x}_q$  (Eq. 9). Then

$$\begin{aligned} \mathbf{Z}_Q \mathbf{x}_Q &= \left( \sum_{q \in Q} \frac{1}{|Q|} \mathbf{Z}_q \right) \left( \sum_{q \in Q} \mathbf{x}_q \right) = \frac{1}{|Q|} \sum_{q \in Q} \left( \mathbf{Z}_q \sum_{r \in Q} \mathbf{x}_r \right) \\ &\approx \frac{1}{|Q|} \sum_{q \in Q} (\kappa \mathbf{Z}_q \mathbf{x}_q) = \frac{\kappa}{|Q|} \sum_{q \in Q} \mathbf{x}_q = \frac{\kappa}{n} \mathbf{x}_Q, \end{aligned} \quad (10)$$

and thus  $\mathbf{x}_Q$  is approximately an eigenvector for  $\mathbf{Z}_Q$ . Since  $\mathbf{x}_Q$  is equivalent to QD-PageRank<sub>Q</sub>, and  $\mathbf{Z}_Q$  describes the behavior of the single-step surfer, QD-PageRank<sub>Q</sub> is approximately the same PageRank that would be obtained by using the single-step surfer. The approximation has the least error when the individual random surfers defined by  $\mathbf{Z}_q$  are very similar or are very dissimilar.

The choice of relevance function  $R_q(j)$  is arbitrary. In the simplest case,  $R_q(j) = R$  is independent of the query term and the document, and QD-PageRank reduces to PageRank. One simple content-dependent function could be  $R_q(j) = 1$  if the term  $q$  appears on page  $j$ , and 0 otherwise. Much more complex functions could be used, such as the well-known TFIDF information retrieval metric, a score obtained by latent semantic indexing, or any heuristic measure using features such as text size and positioning. It is important to note that most current text ranking functions could be easily incorporated into the directed surfer model.

## 6 Scalability

The difficulty with calculating a query-dependent PageRank is that a search engine cannot perform the computation, which can take hours, at query time, when it is

expected to return results in seconds (or less). We surmount this problem by precomputing the individual term rankings QD-PageRank<sub>q</sub> and combining them at query time according to Eq. 9. We show that the computation and storage requirements for QD-PageRank<sub>q</sub> for hundreds of thousands of words is only approximately 100–200 times that of a single query independent PageRank.

Let  $\mathbf{L} = \{q_1, q_2, \dots, q_m\}$  be the set of words in our lexicon. That is, we assume all search queries contain terms in  $\mathbf{L}$ , or we are willing to use plain PageRank for those terms not in  $\mathbf{L}$ . Let  $d_q$  be the number of documents that contain the term  $q$ . Then  $S = \sum_{q \in \mathbf{L}} d_q$  is the number of unique document–term pairs.

## 6.1 Disk Storage

For each term  $q$ , we must store the results of the computation. We add the minor restriction that a search query will only return documents containing all of the terms.<sup>7</sup> Thus, when merging QD-PageRank<sub>q</sub> values, we need only to know the QD-PageRank<sub>q</sub> for documents that contain the term. Each QD-PageRank<sub>q</sub> is a vector of  $d_q$  values. Thus, the space required to store all of the PageRanks is  $S$ , a factor of  $S/N$  times the query independent PageRank alone (recall  $N$  is the number of Web pages). Further, note that the storage space is still considerably less than that required for the search engine’s reverse index, which must store information about all document–term pairs, as opposed to our need to store information about every *unique* document–term pair<sup>8</sup>.

## 6.2 Time Requirements

If  $R_q(j) = 0$  for some document  $j$ , the directed surfer will never arrive at that page. In this case, we know QD-PageRank<sub>q</sub>( $j$ ) = 0, and thus when calculating QD-PageRank<sub>q</sub>, we need only consider the subset of nodes for which  $R_q(j) > 0$ . We add the reasonable constraint that  $R_q(j) = 0$  if term  $q$  does not appear in document  $j$ , which is common for many information retrieval relevance metrics, especially those used by commercial search engines today. The computation for term  $q$  then only needs to consider  $d_q$  documents. Because it is proportional to the number of documents in the graph, the computation of QD-PageRank<sub>q</sub> for all  $q$  in  $\mathbf{W}$  will require  $O(S)$  time, a factor of  $S/N$  times the computation of the query-independent PageRank alone. Furthermore, we have noticed in our experiments that the computation converges in fewer iterations on these smaller subgraphs, empirically reducing the computational requirements to  $0.75 * S/N$ . Additional speed-up may be derived from the fact that for most words, the subgraph will completely fit in memory, unlike PageRank, which (for any large corpus) must repeatedly read the graph structure from disk during computation.

<sup>7</sup> Google has this “feature” as well. See <http://www.google.com/technology/whyuse.html>.

<sup>8</sup> Storing every document–term pair is necessary for certain functionality like term proximity calculation. Most major search engines provide this functionality. If one were to choose not to, it would only need to store every unique document–term pair in its reverse index.

### 6.3 Empirical Scalability

The fraction  $S/N$  is critical to determining the scalability of QD-PageRank. If every document contained vastly different words,  $S/N$  would be proportional to the number of search terms  $m$ . However, this is not the case. Instead, there are a very few words that are found in almost every document, and many words that are found in very few documents;<sup>9</sup> in both cases the contribution to  $S$  is small.

In our database of 1.7 million pages (Sect. 7), we let  $L$  be the set of all unique words (the lexicon), and removed the 100 most common words.<sup>10</sup> This results in  $|L|=2.3$  million words, and the ratio  $S/N$  was found to be 165. Ratio  $S/N$  is essentially the average number of unique words per page, which should remain roughly constant once it has been measured over a large sample of pages. Thus, we expect that  $S/N$  will be approximately 165 even for much larger sets of Web pages. This means QD-PageRank requires approximately 165 times the storage space and 124 times the computation time to allow for arbitrary queries over any of the 2.3 million words (which is still less storage space than is required by the search engine's reverse index alone).

## 7 Results

We give results on two data sets: *educrawl*, and *WebBase*. *Educrawl* is a crawl of the Web restricted to .edu domains. The crawler was seeded with the first 18 results of a search for “university” on Google (www.google.com). Links containing “?” or “cgi-bin” were ignored, and links were only followed if they ended with “.html”. The crawl contains 1.76 million pages over 32,000 different domains. *WebBase* is the first 15 million pages of the Stanford WebBase repository [14], which contains over 120 million pages. For both datasets, HTML tags were removed before processing.

We calculated QD-PageRank as described above, using  $R_q(j)$  = the fraction of words equal to  $q$  in page  $j$ , and  $P(q) = 1/|Q|$ . We compare our algorithm to our implementation of the standard PageRank algorithm. For content ranking, we used the same  $R_q(j)$  function as for QD-PageRank, but, similarly to TFIDF, weighted the contribution of each search term by the log of its inverse document frequency. As there is nothing published about merging PageRank and content rank into one list, the approach we followed was to normalize the two scores and add them. This implicitly assumed that PageRank and content rank were equally important. This resulted in poor PageRank performance, which we found was because the distribution of PageRanks was much more skewed than the distribution of content ranks; normalizing the vectors resulted in PageRank primarily determining the final ranking. To correct this problem, we scaled each vector to have the same average value in its top ten terms before adding the two vectors. This dramatically improved PageRank.

<sup>9</sup> This is because the distribution of words in text tends to follow an inverse power law [21].

We also verified experimentally that the same holds true for the distribution of the number of documents a word is found in.

<sup>10</sup> It is common to remove “stop” words such as *the*, *is*, etc., as they do not affect the search.

**Table 3.** Results on *educrawl*

Query	QD-PR	PR
chinese association	10.75	6.50
computer labs	9.50	13.25
financial aid	8.00	12.38
intramural	16.50	10.25
maternity	12.50	6.75
president office	5.00	11.38
sororities	13.75	7.38
student housing	14.13	10.75
visitor visa	19.25	12.50
<b>Average</b>	<b>12.15</b>	<b>10.13</b>

**Table 4.** Results on *WebBase*

Query	QD-PR	PR
alcoholism	11.50	11.88
architecture	8.45	2.93
bicycling	8.45	6.88
rock climbing	8.43	5.75
Shakespeare	11.53	5.03
stamp collecting	9.13	10.68
vintage car	13.15	8.68
Thailand tourism	16.90	9.75
Zen Buddhism	8.63	10.38
<b>Average</b>	<b>10.68</b>	<b>7.99</b>

For *educrawl*, we requested a single word and two double word search queries from each of three volunteers, resulting in a total of nine queries. For each query, we randomly mixed the top ten results from standard PageRank with the top ten results from QD-PageRank, and gave them to four volunteers, who were asked to rate each search result as a 0 (not relevant), 1 (somewhat relevant, not very good), or 2 (good search result) based on the contents of the page to which it pointed. In Table 3 we present the final rating for each method, per query. This rating was obtained by first summing the ratings for the ten pages from each method for each volunteer, and then averaging the individual ratings. A similar experiment for *WebBase* is given in Table 4. For *WebBase*, we randomly selected the queries from Bharat and Henzinger [2]. The four volunteers for the *WebBase* evaluation were independent from the four for the *educrawl* evaluation, and none knew how the pages they were asked to rate were obtained.

QD-PageRank performs better than PageRank, accomplishing a relative improvement in relevance of 20% on *educrawl* and 34% on *WebBase*. The results are statistically significant ( $p < .03$  for *educrawl* and  $p < .001$  for *WebBase* using a two-tailed paired  $t$ -test, one sample per person per query). Averaging over queries, every vol-

unteer found QD-PageRank to be an improvement over PageRank, though not all differences were statistically significant.

One item to note is that the results on multiple-word queries are not as positive as the results on single-word queries. As discussed in Sect. 5, the combination of single-word QD-PageRanks to calculate the QD-PageRank for a multiple-word query is only an approximation, made for practical reasons. This approximation is worse when the words are highly dependent. Further, some queries, such as “financial aid” have a different intended meaning as a phrase than simply the two words “financial” and “aid.” For queries such as these, the words are highly dependent. We could partially overcome this difficulty by adding the most common phrases to the lexicon, thus treating them the same as single words.

## 8 Open Questions and Future Work

There are several interesting directions to pursue in this area. Initially, search engines used only the contents of a page to measure its quality and relevance to a query. Newer search engines, such as Google, take advantage of the implicit information held in the links between pages. What other sources of information might prove to be useful for future search engines? One remaining source of information is the URL itself. The name of the server, the filename, and the directory structure of the URL are all potentially informative. The IP address of the server may help identify its quality, based on how close it is to a major Internet backbone, who is providing the server, etc. Information about the user him/herself, if accessible, may also be useful (e.g., the user’s bookmark/favorite list contains information about his/her interests, and may be useful in disambiguating queries).

One of the drawbacks of our approach is that, in order retain computational efficiency mentioned, links to pages that do not contain the query term must have zero weight. This may divide the relevant pages into many small, isolated subgraphs. Is there a way that we can allow the surfer to transition through these pages, yet still perform the computation efficiently? What if we always use the same small constant for the probability of transitioning to a page that does not contain the query term? What might the benefits be in combining QD-PageRank with traditional PageRank?

Another approach is to generate QD-PageRanks for clusters of words, instead of for each individual one. For example, we may compute a “pet” QD-PageRank, which prefers pages containing any pet-related term (dog, cat, fish, etc.). To order results for a query about dogs, we would use this pet-specific PageRank. Using this technique, rare words will not face the problem mentioned in the previous paragraph, where pages with the query term are isolated from each other. If we were to use approximately 100 clusters, the computation cost of the calculation would be similar to that of QD-PageRank. Which should we expect to give better results: QD-PageRank for individual terms or for clusters of terms?

Define  $\mathbf{W}(x)$  to be the set of Web pages that contain the term  $x$ . If  $\mathbf{W}(q) \subseteq \mathbf{W}(r)$  then does knowing QD-PageRank <sub>$r$</sub>  assist in the calculation of QD-PageRank <sub>$q$</sub> ? What

if  $\mathbf{W}(q) \subseteq \{\mathbf{W}(r_1) \cup \mathbf{W}(r_2)\}$ ? What if all we know is that  $\frac{\mathbf{W}(q) \cap \mathbf{W}(r)}{\mathbf{W}(q) \cup \mathbf{W}(r)}$  is close to unity (i.e., there is significant overlap between  $\mathbf{W}(q)$  and  $\mathbf{W}(r)$ )?

The  $\text{QD-PageRank}_Q$  for a multiple-word query is a combination of the  $\text{QD-PageRank}_q$  values over  $q \in Q$ . As mentioned, this is only an approximation. Is there a better approximation for  $\text{QD-PageRank}_Q$  that may still be computed quickly at query time? What if, offline, we calculate (and store) cluster-specific  $\text{QD-PageRanks}$ :  $\text{QD-PageRank}_q^c$  where  $q$  is a query term, and  $c$  is one of a small number (100) of query term clusters that represent the other “missing” query terms. At query time, a cluster (or number of clusters) could be chosen, depending on the query, and that subset of  $\text{QD-PageRank}_q$ ’s used in the combination. This may result in a better approximation for  $\text{QD-PageRank}_Q$ , if a suitable definition of  $\text{QD-PageRank}_q^c$  is found. Jeh and Windom’s work [16] may also be applicable here.

## 9 Conclusions

In this chapter we explored the idea that links carry useful information for Web search. We introduced the two most common algorithms for extracting this information, HITS and PageRank, and discussed some of the drawbacks of each. In particular, HITS is not efficient enough at query time, and PageRank (and, to a lesser extent HITS) suffers from topic drift. We then introduced a model that overcomes these problems by probabilistically combining page content and link structure in the form of an intelligent random surfer. The model accommodates most query relevance functions in use today and produces higher-quality results than PageRank, while having time and storage requirements that are within reason for today’s large-scale search engines.

## 10 Acknowledgements

We would like to thank Gary Wesley and Taher Haveliwala for their help with WebBase, Frank McSherry for eigen-help, and our experiment volunteers for their time. This work was partially supported by NSF CAREER and IBM Faculty awards to the second author.

## References

1. K. Bharat, A.Z. Broder, M.R. Henzinger, P. Kumar, and S. Venkatasubramanian. The connectivity server: Fast access to linkage information on the Web. *WWW7 / Computer Networks*, 30(1-7):469–477, 1998.
2. K. Bharat and M.R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the Twenty-First Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, 1998. ACM Press.

3. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the Seventh International World Wide Web Conference*, pages 107–117, Brisbane, Australia, 1998.
4. S. Chakrabarti. Data mining for hypertext: A tutorial survey. *SIGKDD Explorations*, 1(2):1–11, 2000.
5. S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann, San Francisco, Ca., 2003.
6. S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Mining the Web's link structure. *IEEE Computer*, August 1999.
7. S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource compilation by analyzing hyperlink structure and associated text. In *Proceedings of the Seventh International World Wide Web Conference*, pages 65–74, Brisbane, Australia, 1998. Elsevier.
8. S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 307–318, Seattle, WA, 1998. ACM Press.
9. D. Cohn and H. Chang. Learning to probabilistically identify authoritative documents. In *Proc. 17th International Conf. on Machine Learning*, pages 167–174. Morgan Kaufmann, San Francisco, CA, 2000.
10. D. Cohn and T. Hofmann. The missing link – a probabilistic model of document content and hypertext connectivity. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*. MIT Press, Cambridge, MA, 2001.
11. B.D. Davison. Topical locality in the Web. In *Twenty-third Annual International Conference on Research and Development in Information Retrieval*, pages 272–279, Athens, Greece, 2000.
12. G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.
13. T. Haveliwala. Topic-sensitive PageRank. In *Proceedings of the 11th International World Wide Web Conference*, Honolulu, HI, USA, May 2002.
14. J. Hirai, S. Raghavan, H. Garcia-Molina, and Andreas Paepcke. WebBase: A repository of Web pages. In *Proceedings of the 9th International World Wide Web Conference (WWW9)*, 2000.
15. T. Hofmann. Probabilistic latent semantic analysis. In *Proc. of Uncertainty in Artificial Intelligence, UAI'99*, Stockholm, 1999.
16. G. Jeh and J. Windom. Scaling personalized Web search. In *Proceedings of the Twelfth International World Wide Web Conference*, 2003.
17. J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
18. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford University, Stanford, CA, 1998.
19. M. Richardson and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in PageRank. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 1441–1448. MIT Press, Cambridge, MA, 2002.
20. G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY, 1983.
21. G.K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, Cambridge, MA, 1949.



---

# Search Engine Ability to Cope With the Changing Web

Judit Bar-Ilan<sup>1</sup>

The Hebrew University of Jerusalem,  
Jerusalem, Israel

judit@cc.huji.ac.il

**Summary.** This chapter discusses Web search engine performance over time. Unlike classical information retrieval systems, the Web is decentralized and dynamic, that is, new pages are added, others are moved and removed, while existing pages may undergo changes. The dynamic nature of Web pages should influence search engine results over time. A set of measures is introduced to evaluate search engine performance in this constantly changing environment. The chapter also discusses in short search engine architecture, models and characterizations on the growing and changing Web, and reviews a number of small experiences that demonstrate that search engines do not always cope satisfactorily with dynamic changes.

## 1 Introduction

The World Wide Web has become a major information source. The Web is huge, and it is growing constantly. Because of its size it is almost impossible to find our way in this “information chaos” without “finding aids”. Thus the Web can be viewed as a huge information retrieval (IR) system, where the documents are the Web pages and the general-purpose automatic search engines are the retrieval tools. This analogy allowed for the early development of search tools on the Web. The first systems relied heavily on experience based on years of research concerning traditional IR systems.

The Web, however, is very different from traditional IR systems in several aspects. Traditional systems operate in a highly controlled, centralized and relatively stable environment. New documents are added to these systems in a controlled fashion. Sometimes old documents are removed or moved (for example, in case the current database of a bibliographic retrieval system contains only documents from the last two years, and the older ones are moved to an archive); and occasionally documents or document representations change so mistakes can be corrected. The major point is that all these processes are controlled.

The Web, on the other hand, is uncontrolled, distributed and highly dynamic; it is in a state of constant dynamic flux. The situation was aptly expressed by Chakrabarti et al. [17]: “the Web has evolved into a global mess of previously unimagined proportions”. On the Web almost anyone can publish almost anything. Later the author of the page

or the publishing site may decide to change the content, to remove the page or to move it to another directory on the same server or to move it to a different server, or to publish the same content at another URL. All these changes occur continuously, and the search engines are not notified of any of them. They not only have to cope with the dynamic changes caused by the authors and the publishers but also with failures of servers and communication lines. Search engines also have to fight spammers, who invent all sorts of tricks in order to raise the rankings of their pages on certain queries (for example, see [30]). The rate of growth of the Web [35, 36, 41] is also unparalleled to the rate of growth of traditional IR systems.

Thus search engines operate in a highly dynamic and sometimes hostile environment - rather different from the controlled and centralized environments in which traditional IR systems function. Still, most of the evaluation of search engine performance is based on traditional IR evaluation criteria. The main topic in this chapter (Sect. 6) is to develop tools for evaluating search engine performance under these circumstances. These tools allow us to assess whether or not they add another level of instability to the already dynamic situation. The chapter also discusses prevailing search engine architectures (Sect. 2), evaluation of search engine performance based on traditional criteria (Sect. 3), works studying the dynamic nature of the Web and introducing refresh and recrawl techniques (Sect. 4); and some works exhibiting problems with search engine performance in dynamic situations (Sect.5).

## 2 Search Engine Architecture

In the context of this paper *search engines* are defined as general-purpose tools, that visit, index and retrieve Web pages automatically (e.g., [1]). Sometimes directory services (e.g., the Open Directory ([www.dmoz.org](http://www.dmoz.org)), or the human-powered part of Yahoo! ([www.yahoo.com](http://www.yahoo.com))) are also considered search engines [49]. Even though some of the issues discussed are also relevant to directory services, here we concentrate of *crawler-based* search engines, like Google ([www.google.com](http://www.google.com)), Fast ([www.alltheweb.com](http://www.alltheweb.com)), AltaVista ([www.altavista.com](http://www.altavista.com)), Inktomi (through one of its interfaces, e.g., MSN search ([www.search.msn.com](http://www.search.msn.com)) or HotBot ([www.hotbot.com](http://www.hotbot.com))) or Teoma ([www.teoma.com](http://www.teoma.com)). Note that search engine alliances and ownerships undergo frequent changes. We listed here the major crawler-based search engines as of early 2003. Searchengine Watch ([www.searchenginewatch.com](http://www.searchenginewatch.com)) is a good place to look for current information on search engines.

A crawler-based or automatic search engine is made up of three major parts: the crawler, the indexer and the query engine (for further details, see [3, 13, 4, 42]). Now we describe the main functions of each of these parts.

The crawler is responsible for data collection. It is a software utility, which visits URLs (the addresses of the Web pages) and forwards their contents to the indexer. Usually a number of crawlers run in parallel, each with a different “seed” (an initial set of URLs to be visited). The crawler constantly updates the set of URLs to be visited by extracting links from the previously seen pages. Each search engine has a policy on which pages to visit, in which order, and how often to revisit them (for further

details regarding crawling policies, consult the chapter by Pant et al. on “Crawling the Web”). The differences in the respective policies are the main causes of differences between the search engines in terms of size, coverage, and freshness. Revisits are necessary since Web pages, unlike printed documents, may undergo changes and/or disappear altogether.

The indexer extracts words from the pages forwarded to it by the crawlers and creates a list of words that appear in the document. Usually for each word in the list its locations on the page and some additional characteristics (e.g., font, type of emphasis, type of HTML tag in which it appears) are also recorded. The pairs (URL, list of words on page) are inverted into pairs: (word, list of URLs in which the word appears). This data structure is called the “inverted index” or the “inverted file”, and it facilitates the work of the query engine. Search engines may also work with other data structures, but the inverted file method is most prevalent. The search engine may also decide to save extra information about the document (e.g., metadata describing its contents, the size of the page, its date of creation/update) or about the linkage structure of the crawled pages. Most search engines discard the copies of the visited Web pages (a well-known exception is Google, which allows users to view a copy of the page as was seen by the crawler, using the “cached” feature), and base their retrieval only on the inverted index and the additional information recorded by them.

The query engine is responsible for receiving user requests and for providing answers to these requests. When receiving a query, the query engine consults the inverted file and the database of additional information. Search engines differ in the search capabilities they provide (e.g., phrase searching, Boolean expressions, or limiting the search to certain domains, geographic areas, or time spans). For most queries the number of documents related to the query is enormous; thus the search engines display ranked lists of results, with the most relevant results (in their opinion) displayed at the top of the list. They apply different ranking algorithms, including classical ones based on similarity between the document and the query (a function of the frequency of the search terms in the document and the rarity of the search terms in the database), the location of the search terms in the document (terms in title or headers receive more weight), and algorithms that are applicable to hypertext systems (e.g., link analysis: the rank of a page is a function of the number of links to it from other pages on the Web, where links from more “important” pages are given higher weight; or “clickthrough”: how often a page is visited, as observed by an objective service). Each search engine uses a different, secret “recipe” to compute the ranks. The ranking algorithms are not revealed for two reasons: they are trade secrets, and not to disclose information to potential spammers. Search engines also differ in the ways the search results are displayed (e.g., summaries, number of results per page, clustering or classification of results) and in additional search-related features provided by them (e.g., refine search, suggested search terms, “more results like this”, spell checking).

Most search engines have several servers or clusters of servers, and the actual architecture is much more distributed than the basic architecture detailed here. Risvik and Michelsen [42] from the search engine FAST discuss how large-scale search engines handle growth and dynamics on a Web.

A different, highly distributed architecture was developed by the Harvest team [11]. The Harvest system is made up of a series of Gatherers (which roughly correspond to crawlers) and Brokers (which provide an indexed query interface) and Glimpse (a search interface); additional interfaces were also developed. The basic idea was to utilize existing computing and search power on Web servers, with machines working together to handle loads that a single machine could not handle. The Harvest code is free, and even though the original developers are no longer involved, it is still maintained and available (as of the beginning of 2003) from <http://harvest.sourceforge.net>. One of the goals of the Harvest system was to be highly scalable, however, it has its limitations regarding both the size of the indexes and the speed of the retrieval [28].

### 3 Search Engine Evaluation

In this section we mention a few works related to the evaluation of search engines. There are many studies in this area. Here we only mention a few to outline the different criteria used so far for the evaluation of search engine performance.

The two best-known efficiency measures for IR systems are *precision* and *recall*. *Precision* measures the percentage of *relevant* retrieved documents out of the total number of retrieved documents for a given query, and *recall* measures the percentage of *relevant* retrieved documents out of the total number of documents that are *relevant* to the query and are stored (or indexed by) the system. *Relevance* is an extremely subjective notion, and is much discussed by the information science community [44, 39]. Even though *precision* and *recall* are based on the highly questionable notion of relevance, they are still very frequently used to evaluate IR systems. Computing *recall* poses additional problems, since it is very difficult even to estimate the number of relevant documents (even if we have an operational definition of relevance) to a query that exist in the system. It is very interesting to note that *precision* and *recall* were among the six criteria introduced by Cleverdon in 1964 [21] when working on the Cranfield project. All of these criteria are still widely used, in spite of the enormous technological developments in the past 40 years. The criteria are:

- response time
- coverage
- user friendliness
- form of output
- precision
- recall

Initial evaluations of Web search engines mainly concentrated on the *precision* of the different tools. Usually only the relevance of the first 10 or 20 results was judged. This choice is quite reasonable, since users usually do not go beyond the first 10 or 20 results [46, 48]. Chu and Rosenthal [20] evaluated three search engines using the classical measures introduced by Cleverdon, with an emphasis on precision. Two judges evaluated the relevance of the first ten results for ten sample queries. Ding

and Marchionini [24] also evaluated the relevance of the first ten results for four sample queries. They judged relevance on a six-point scale and specified the scale for each query separately. Leighton and Srivastava [37] used a five-point scale, but the relevance levels were not query specific. In all these cases the researchers and not the actual users judged the queries. As in traditional IR, the question who should be the judge of the relevance of documents is much debated. Gordon and Pathak [27] claim that relevance judgments can only be carried out by the users who posed the queries; while others [29] challenged this approach, since it clearly limits the scope of evaluations.

On the Web *recall* is rarely calculated because in the definition of *recall*, the number of relevant pages indexed by the system is often and mistakenly substituted by the number of relevant pages existing on the Web. Since it is well-known that no search tool covers the entire Web, [10, 35, 36], most researchers conclude that *recall* on the Web cannot be measured. The importance of *recall* as a measure for search effectiveness was already questioned by Cleverdon for traditional IR systems [22]. The utility of this measure becomes even more questionable on the Web [15] where for all but the most obscure queries there are thousands of documents containing the search terms but most users do not go beyond the first ten or twenty search results [48]. Queries on the Web are substantially different from queries presented to traditional IR systems. Broder [16] discussed this issue and proposed a taxonomy of Web queries. He defined three classes: navigational (looking for a site of an entity), informational (acquire some information, matches most closely the queries in traditional IR systems), and transactional (where the query initiates some interaction, and the results of the query usually cannot be evaluated on spot. As an example, consider e-shopping for an item, where the user can decide whether he is satisfied or not only after he receives the item.).

Besides *precision*, the *coverage* of search engines was also computed. In classical systems, *coverage* means how extensively an IR system covers a given subject matter. On the Web for general search engines, the parallel question is how many pages are indexed by the search engines and what is the overlap between them. At the “beginning of the Web”, search engines were very proud of their coverage. Lycos, for example, claimed to have indexed 91% of the Web [53]. Impartial and empirical results indicated that this was not the case [10, 35, 36]. In fact, the search engines covered only a small portion of the “indexable Web” (which is only a small portion of the Web pages freely accessible to Web surfers), and the overlap between the search engines was also very small at the time these studies were carried out.

Besides the above-mentioned criteria, other evaluation measures were used as well. In an early article, Courtois et al. [23] evaluated search tools based on “known-item” searches (i.e., they were looking for specific documents). This method is most applicable for evaluating navigational queries using Broder’s terminology. Hawking et al. (1999) found that the Text REtrieval Conference (TREC, <http://trec.nist.gov>) systems were superior to Web search engines in terms of first twenty precision. The comparison had its limitations, since the TREC systems ran on TREC’s 18.5 million pages test collection, while the search engines used their own (substantially larger) databases, and the sample queries came from the TREC query set. Web queries,

however, are often rather different from queries presented to traditional IR systems, and one of the reasons for the superiority of TREC systems could have been that these systems were targeted toward such queries. The findings of Singhal and Kaszkiel [47] support this claim: when they compared a state-of-the-art TREC system with Web search engines looking for home pages of entities (navigational queries), the results of the search engines were superior to the results of the TREC system. A search was considered successful if the home page of the entity was among the top-ten pages retrieved for a search. The search tools were credited according to the rank of the home page among the top-ten results retrieved. Zhu and Gauch [54] experimented with “quality metrics” (e.g., authority, availability, popularity) for ranking and evaluation.

Search engines cannot be evaluated without understanding the needs and attitudes of users. Silverstein et al. [46] and Spink et al. [48] tried to characterize users and their queries through analyzing large query logs. Holscher and Strube [31] created a model of information seeking on the Web based on interviews with “Internet experts”. This model was then tested on a group of experts and novices. Watson [51] interviewed nine eighth-grade students in order to try to understand their perceptions of the Web and Web searching.

Note that none of the above-mentioned works took into account the dynamic nature of the Web and did not examine the performance of the search engines under these circumstances. In the next section we review the literature related to the dynamic nature of the Web.

## 4 The Dynamic Web

### 4.1 Modeling the Web

There is great interest in understanding the structure of the Web. The Web can be viewed as a directed graph where the Web pages are its nodes and the hypertext links between pages are its directed edges. Albert et al. [2] found, based on experimental results, that both the number of incoming links to a page and the number of outgoing links follow power laws. Such findings can be explained by well-defined growth models. Broder et al. [14] gave a detailed view of the Web linkage structure, deduced from a 200-million page crawl of AltaVista. According to their findings the Web graph is made up of four parts: the strongly connected component (SCC), IN, OUT, and the TENDRILS. In the strongly connected center (SCC), there exists a directed path between every pair of nodes belonging to this set. That is, beginning from any node  $a$  we can reach any node  $b$  (both in SCC) by clicking on a specific link on each of the intermediate pages. For the IN part of the Web graph, there exists a directed path from every node  $a$  belonging to this set to every node in SCC, but there is no such path from any of the nodes of SCC to  $a$ . For every node  $b$  in the subgraph OUT there exists a directed path from every node in SCC to  $b$ , but there is no directed path from  $b$  to SCC. The rest of the nodes belong to a set called TENDRILS: for nodes that belong to this set, there is no directed paths to or from the SCC. All sets are of roughly equal size. This structure can explain the low overlap between search engines: nodes

in IN and in the TENDRILS are reached by crawlers only if they or some other node in the same set appear in the set of initial URLs for the crawler. Broder et al. also confirmed that the number of in-links and out-links follow power laws.

Based on empirical findings, Barabasi and Albert [9] modeled the Web as a scale-free network. Their model is based on growth and preferential attachment, which explains how such structures are created and fit the empirical results. The model takes into account that the system is built over a period of time and at each step new nodes are created. The new node is connected to existing nodes, where the probability of being linked to a given node is proportional to the number of links already pointing to that node. This model totally ignores the fact that on the Web not only are new nodes created, but existing ones may disappear. In addition, existing nodes sometimes undergo changes, a process that may alter the linkage to and from this page. Authors of pages who have not even considered linking to the page in its previous form may choose to link to it now, while other authors who linked to the page before may choose to remove their links. Since authors of pages with links to a given target are usually not notified of changes in the content of the target, the process of changing the in-links to the target may take a long time. (Web authors often cease to maintain their pages, thus the removal of the links because of a change of content may possibly never be completed.)

Even though the growth and preferential attachment model ignores disappearance and modifications, it still models the Web quite accurately. So maybe disappearance and changes are not that frequent after all? A number of studies tried to answer this question.

## 4.2 Caching and Update Algorithms Under Dynamic Changes

Douglis et al. [25] studied the rate of change of Web pages, with the aim to assess the usefulness of caching. They analyzed a log of 17 days of Web access from AT&T Labs. Of the pages accessed more than once during the period, only 13% were to a resource that had been modified since the previous access.

Brewington and Cybenko [12] observed rate of change through a Web service that downloaded about 100,000 pages per day for a period of 7 months. Only 4% of the pages were totally dynamic (different each time they were accessed), while 56% of the pages were completely stable during the observation period. Brewington and Cybenko introduced a metric of “up-to-datedness” in order to estimate the rate at which Web search engines should reindex the pages they cover in order to remain current. The metric,  $(\alpha, \beta)$ -currency, measures “up-to-datedness”, where  $\alpha$  is the probability that the page indexed by the search engine is  $\beta$ -current. That is, had we compared the copy the search engine had with the copy on the Web not less than  $\beta$  time units ago, we would have seen identical copies.  $\beta$  is called the “grace- period”, which allows users to “forgive” changes that have occurred within  $\beta$  time of the present. They modeled the growth of the Web as an exponential process, where the exponential distribution incorporates two processes: creation of new pages and modification of existing ones, and the exponent is both a function of the growth rate and of the time since the last

modification of the page. This model takes into account modifications but still ignores the rate of disappearance of pages.

Cho and Garcia-Molina [19] introduced and compared several policies for updating local databases. Their framework is based on two metrics, *freshness* and *age*, where *freshness* counts the number of pages in the local database that are identical to page on the Web at a given time, and *age* of a page is 0 if it the local copy is the same as the copy on the Web at time  $t$ , and otherwise it is  $(t - \text{modification time of the page})$ . In [18] they present the findings of a large-scale experiment, in which they observed the daily changes to more than half million pages for a period of 4 months (3000 pages from each of the 132 selected popular sites). On the one hand, more than 20% of the observed pages changed daily, but on the other hand, about 30% of the pages had not changed at all. They also tried to estimate the “lifespan” of Web pages: about 16–19% of the pages exist on the Web for more than a week but for less than a month, and between 35% to 50% exist for more than 4 months. Their calculations showed that it took about 50 days for 50% of the pages to be changed. Pages were modified at a much faster pace in the “.com” domain: it only took 11 days in that domain for 50% of the pages to be modified. Note that the changes occurring to pages are not characterized, these could have been dynamic elements, e.g. the date, that change each time the page is downloaded.

Risvik and Michelson [42] discussed the implications of growth and Web dynamics for search engines. They show how the search engine FAST handles the problems arising from the growth, the disappearance, and the modification of pages. Their data indicate that pages are updated every 11 days at a rate of 400 documents per second, and newly discovered pages are added to the database only in place of pages that are deleted from the repository “because the crawler discovers that they have been deleted from the Web or for other reasons” (which are not specified). The rate of update may not be entirely true, as a specific example, as of March 1, 2003, FAST indexed 333 pages from the server ms161u06.u-3mrs.fr; this server, however, has not responded to our repeated requests since April 22, 2001. Of course, this single example could be an exception to the general rule for refreshing pages.

### 4.3 Characterizing Changing Web Pages

The above-mentioned papers studied the rates of change for technical reasons, mainly for devising better caching or update methodologies. Other papers examined modifications to Web pages in order to characterize these changes.

Koehler, in two papers ([33] and [34]) described the results of a four-year long experiment of the weekly retrieval of 321 Web pages. These Web pages were generated using the no-longer existing WebCrawler random URL and generator and were supplemented (at least according to the first paper) with additional URLs from HotBot in order to reflect the reported distribution of Web pages by top-level as of December 1996. This set of URLs was fixed for the duration of the experiment. Results show that after four years 34.4% of the URLs still existed, but even after one year only 11 pages (3%) have not been modified in any way. The rest of the pages were modified during the period; they had undergone either content or structural changes.



Even though pages are being modified, for the majority of the pages, changes were observed in less than 15% of the observation points during the first year. Over the four year period Koehler observed a decline in content changes and an increase in structural (linkage) changes.

A different approach to characterizing changes on the Web was taken by Bar-Ilan and Peritz [8]. Instead of trying to create a “random” set of URLs or examining a set of popular sites, they collected pages on a specific topic, “informetrics”. In order to be as exhaustive as possible at each data collection point, the six largest search engines at the time were queried. Data were collected once a month over a period of six months at the beginning of 1998, and a follow-up data collection was carried out in June 1999. Out of the retrieved pages, only pages on the scientific topic were retained. This method of data collection enabled us not only to observe modifications and disappearance of pages, but also the appearance of new pages on the topic (as recorded by the search engines). Overall a moderate growth trend was observed (711 URLs at the first data collection point, and 806 URLs at the last). The chosen topic turned out to be very stable; 47.2% of the URLs were retrieved by the search engines at each of the six initial data collection points. Most of the pages that were retrieved more than once had not undergone any changes. Only 34.4% of the URLs changed during the observation period. These changes were characterized as minor/major and stagnant/dynamic (the number of times the page changed). The results showed that if a page is modified, it is likely to be modified frequently and substantially.

In a most recent study by Fetterly et al. [26] more than 150 million pages were revisited once a week for eleven weeks. The changes between every consecutive pair of successful downloads were analyzed, and the findings show that there was no change in 62.5% of the pairs. The average degree of change varied across top-level domains (the highest rate of change was observed for Germany – .de, and the lowest rate for .edu and .cn – China), and larger pages changed more often and more severely than smaller ones.

## 5 Search Engines in a Dynamic Environment

In this section we review empirical works related to search engine performance over time. We were able to locate only a small number of such works, and most of them follow the changes in the number of results for only a very few queries. Thus the observed changes may not be representative of the search engine’s behaviour as a whole. Search engine algorithms and technologies change constantly, and therefore the results of an experiment carried out at a given time may not hold or be relevant at some other time. However, the observed fluctuations demonstrate the need for developing metrics to evaluate the performance of search engines in dynamic environments.

We [5] studied the search results of the query “informetrics OR informetric”, for a period of five months in 1998, using the six largest search engines at the time (AltaVista, Excite, HotBot, Infoseek, Lycos, and Northern Light). The search engines were queried once a month, and all the search results were downloaded. To our great surprise we found that sometimes existing pages on the topic disappeared from the

search results, and occasionally they reappeared again at some later point in time. This behavior was most notable for Excite, which “forgot” 72% of the technically relevant URLs it retrieved. In each of the search rounds Excite retrieved almost the same number of results (158 URLs on the average), but when we compared the sets of URLs we were rather surprised to discover that the overlap between the sets was very small – during the whole search period Excite retrieved a total of 535 technically relevant URLs. This result shows that the number of results only (without checking the list of URLs) may hide the fact that there are fluctuations in the results.

At about the same time, Rousseau [43] submitted three single-word queries (trumpet\*, saxophone\*, and pope) daily to the search engines AltaVista and Northern Light for a period of 21 weeks. In this study, only the numbers of results were recorded. Northern Light was very stable during the whole period, exhibiting a small monotonic rise in the number of results, as could be expected from the growth trend on the Web. On the other hand, the number of results reported for AltaVista fluctuated greatly, and sometimes daily jumps of 20% or more were observed. Except for the day the “new AltaVista” was introduced, these fluctuations could not be explained. When noise reduction techniques were applied (the five-day median was computed instead of the daily results) the graph became much smoother. Rousseau’s recommendation was not to use AltaVista for research purposes unless one needed a unique feature of this search engine. AltaVista was (and maybe is) the most popular search engine for research purposes because of its specific features. For example, Kleinberg [32] drew the initial data sets on which the hubs and authorities algorithm was run from AltaVista.

Rousseau was not the only one to observe daily fluctuations in search results. We [6] submitted 20 queries to the search engines HotBot and Snap, daily for a period of 10 days in October 1999. Huge fluctuations were observed for HotBot, while the results for Snap were very stable. At that time the search results of both services were drawn from the Inktomi database. Thus the results indicated that the fluctuations were probably not caused by the database, but by the search interface of HotBot.

Selberg and Etzioni [45] submitted 25 queries (chosen from the set of queries used by Lawrence and Giles [35] to estimate the size of the Web) 25 times (the intervals between searches were not constant) over a one month period to 9 search tools. They calculated the set difference between the top 10 and top 200 search results of each of two consecutive searches. They expected to observe some changes because URLs are constantly being added and removed, but they were surprised at the extent of instability.

Mettrop and Nieuwenhuysen [38] also analyzed fluctuations in search engine results. They introduced a new methodology to study the problem. They created a document and published it on 16 different URLs. Some of the URLs were submitted to the search engines and linked to a URL indexed by the search engines. They formulated 32 queries relating to different aspects of the document; and ran these queries on 13 search tools for over 40 times during a 15-month period in 1998–1999. Their findings showed that the result sets not only fluctuated over time, but at a given time a search engine retrieved a URL for some query formulations but not for others.

What are possible reasons for these fluctuations?

- Search engine resources are limited, both in terms of space and crawling and indexing capability, thus their size is more or less constant (at least for a period of time), and in the process of adding new URLs to their database, they delete existing ones [5, 42].
- Some of the search engines have several query engines or databases; each time the query is routed to a different server, and the query engines are not identical, thus there are differences in the results (see [38] in which the explanations are based on an interview with Andrei Broder, then chief scientist of AltaVista, or [52] on the different Google servers).
- The index is partitioned (known to be so for FAST [42] and for Inktomi), and if some of the partitions are down or slow, the search engine returns partial results [6].
- When the crawler refreshes its database, some of the previously visited pages may be unreachable due to communication or server failures (not of the search engine, but of the server on which the page resides) [6].
- If the search engine uses shadowing (see [3]) it has a database that serves the queries and another database that is being built based upon the current crawling; the “new” database replaces the old one at some point in time, and it is possible that the new database covers a substantially different set from the old one.
- Search engines trade off quality for speed and limit the resources available for each query (timeouts). Such behavior was experienced by Selberg and Etzioni [45]. They call this technique “thresholding”. In all our experiments we carried out our searches at light load times for the search services, still it is possible that the results of the query are not based on the whole index [5].
- Search engines use performance-dependent algorithms, and when they are busy these algorithms are not always activated, which may cause fluctuations. This point appears in [38] but is not elaborated.
- Search engines attempt to remove duplicates, and thus may retain only a single copy of multiple URLs with identical content [38]. If this is the cause for removing pages, then it is not clear why they reappear later. For a specific query [7] we tested whether this could be the case, and our findings greatly ruled out this possibility.
- There could be errors in the indexing or in the retrieving processes. This may explain why, when trying to locate a specific page through different query formulations, the page is retrieved for some of the formulations and not for others [38].
- Search engines may deliberately drop pages that are not clicked at. Sullivan [50] reported that Inktomi removed pages that did not seem to satisfy search queries. It is not clear why such pages are included again.
- With the prevalence of “paid inclusion” programs, pages that are not included in these programs (mainly pages from noncommercial domains) may be candidates for removal. They may be included again if the search engine has enough resources left, after taking care of its commitments to the “paid inclusion” sites.

- Fluctuations may be due to changes in the indexing policy of the search engines or in the size of the database. This was the case with AltaVista in [43] and with Google in [7].
- Search engines frequently change their business models and strategies, which may have influence on the search results. At this moment ownerships and alliances are in a flux, however, this was not so much the case when the reviewed studies took place.
- When considering top results only (as in [45]) changes in the ranking algorithms may also explain some fluctuations. Most ranking algorithms incorporate link analysis (e.g., [13]), thus changes observed in the link structure of the Web by search engines may have influence on the set of top 10 or top 200 results.

Lawrence and Giles [36] raised an interesting point about search engines: “There may be a point beyond which it is not economical for them to improve their coverage and timeliness.” Thus, in addition to the technical and algorithmic difficulties, the financial aspects must also be taken into account. We are not only facing the question whether the search engines can cope with the growth and changes on the Web, but also whether they want to cope. The commercial objectives of search engines will probably have increasingly high priorities over time.

We enumerated a number of possible causes for fluctuations in search engine results, and we may have missed some additional reasons. Whatever causes this behavior, the authors of the papers alerting to these fluctuations seem to agree that this causes damage to the user’s search experience and his/her ability to replicate results. Sources that they usually locate with the help of the search engines cannot be found through them anymore (sometimes only temporarily), leaving the user under the impression that the source disappeared from the Web. The search engines (at least some of them, some of the time) add another level of instability to the already chaotic Web. In order to study this behavior, and not just to alert users to strange search results, appropriate evaluation criteria have to be developed. The major issues of interest when coming to evaluate search engine performance in a dynamic environment are:

#### 1. Timeliness/freshness

- Percentage of broken links
- Percentage of pages where the indexed copy differs from the Web copy
- Percentage of “recently” created pages in the database

#### 2. Stability over time

- Are there great fluctuations in the number results for a given query?
- Does the search engine “drop” from its database existing URLs relevant to the query?

In the next section we describe the necessary framework for evaluation, then we formally define the metrics and discuss their meaning. We also review the results of a small experiment for which these metrics were computed.

## 6 Evaluation of Search Performance in a Dynamic Environment

### 6.1 The Framework

This framework for evaluating search engine performance over time is based on [7]. In order to evaluate search engine performance over a period of time, the query or queries have to be submitted periodically to the search engine. The query/queries are run in *search rounds* (*rounds*, for short). The search period is the span of time during which the searches were carried out. The search rounds should be equidistant. From our experience it is sufficient to run the query/queries once a month. We experienced with running the query once a week, but the observed changes were not very significant. An exception was an experiment we carried out in September–October 1999, when huge daily fluctuations were observed in the results of HotBot [6]. Notess [40] reports that AltaVista has an ongoing problem with the number of results: because of unreported timeouts, it may retrieve a different number of results each time the search button is pressed. We have not encountered such problems with AltaVista during our searches. However, we made sure that the queries were run at a time when Internet communication is known to be low, on Sunday early mornings (around 5:00–7:00 GMT).

In order to compute the percentage of newly added pages, the percentage of “dropped” pages, and the percentage of broken links, all the URLs the search results point to must be visited in every search round. The best solution is to download all the pages the search results point to immediately after the query is run. This way the results can be examined in a more leisurely fashion, and what is more important, they can be viewed as they were seen at the time the searches were carried out by anyone wishing to inspect the results at a later time. The above requirement restricts the queries on which the search engine can be evaluated. The entire set of search results must be examined, thus the query has to be such that the search engine presents all the hits for the given query. Most search engines limit the number of displayed results – they usually do not display more than the first 1000 results. (AltaVista displayed only 200 at the time of the specific experiment, but this problem can be partially solved by carrying out several searches limited to different dates of creation of the URLs.) Further steps must be taken in order to retrieve all the hits for search engines that cluster the search results.

We also need a method to decide which of the retrieved documents are “relevant” to the query. As we mentioned before (Sect. 3), relevance is a very difficult notion, and has been heavily discussed by the IR community, yet there is no general agreement on how to judge relevance. Human relevance judgment in the case of periodic searches with a large number of results is a repetitive and time-consuming task, therefore it is not feasible. Thus we defined a more lenient measure, called *technical relevance* that can be computed automatically. A document is defined to be *technically relevant* (in the following we shall use TR as shorthand for *technically relevant*) if it fulfills all the conditions posed by the query: all search terms and phrases that suppose to appear in the document do appear, and all terms and phrases that are supposed to be missing from the document, terms preceded by a minus sign or a NOT operator, do not

appear in the document. A URL is called a *technically relevant URL* if it contains a technically relevant document. Lawrence and Giles also took this approach [36] even though they point out “search engines can return documents that do not contain the query terms (for example documents with morphological variants or related terms)”. It is advisable to choose query terms with as few morphological variants as possible. (Northern Light, for example, did not differentiate between pages in which the query term appears in singular or in plural in cases of simple plural.) From our experience, related terms or concepts are very rarely substituted for the original query terms.

Unlike the classical notion of relevance (based on human judgment), *technical relevance* is objective and reproducible. On the other hand, *technical relevance* cannot differentiate between documents providing extensive and useful information on the search topic and between documents in which the search topic is mentioned only superficially, e.g., in a sidebar of the Web page. Evaluating *technical relevance* provides a fast and easy method to differentiate between pages “about” the search topic and pages that clearly have nothing to do with the query (including broken links and otherwise inaccessible pages), but we must also be aware of its limitations.

## 6.2 Defining the Metrics

To evaluate the percentage of broken links, we define:

$$broken(q, i) = (\# \text{ broken links}) / (\# \text{ results for query } q \text{ in round } i).$$

There are temporary communication failures, which may result in 404 messages, thus a second attempt must be made (at a slightly later time) to download these pages before deciding that they are really missing or inaccessible. Next we introduce *new*, which counts the number of newly added URLs for  $i > 1$ :

$$new(q, i) = |\{\text{TR URLs retrieved in round } i\} \setminus \{\text{TR URLs retrieved in round } j \text{ where } j < i\}|.$$

This measure is influenced both by the growth of the subject on the Web and by the rate at which the search engine adds new pages to its database. A new page may be added to the search engine’s database for two reasons. First, because the page was created recently or its content was recently changed, so that the page became relevant for the query. Second, because the page had already existed and had been relevant to the query for a “long” time, but the search engine only recently discovered it and added it to its database.

In order to try to differentiate between the two factors influencing *new*, we may run the same query on several large search engines in parallel, and try to create an “exhaustive” pool of pages *technically relevant* to the query for each search round. If more than one search engine is used for data collection then  $new(q, i)$  counts the number of URLs that have not been located by any of the search engines before round  $i$ . Then we can partition  $new(q, i)$  into  $totally\_new(q, i, s)$ , these are URLs that are discovered for the first time in round  $i$  by any of the search engines, and

$newly\_discovered(q, i, s)$ , these are URLs that were already discovered by other search engines in one of the previous rounds, but appear for the first time in the list of URLs located by  $s$ .

$$totally\_new(q, i, s) = |\{\text{TR URLs retrieved by } s \text{ in round } i\} \setminus \{\text{URLs in the pool of TR URLs retrieved before round } i\}|,$$

$$newly\_discovered(q, i, s) = new(q, i) - totally\_new(q, i, s).$$

There are no easy means to decide whether the search engine's information is outdated, except, perhaps, in case the document is totally unrelated to the query (not even the same concept). Some partial conclusions may be drawn from the search engine's summaries. An exception is Google, which caches most of the URLs it visits; thus is possible to compare the downloaded pages with the cached versions. In order to carry out this comparison, the cached documents should also be downloaded, in order to compare the local and the Web copy, as they existed at the given point in time. These suggestions are "work-arounds", and therefore we have not defined a measure to evaluate the extent to which the search engine's information is outdated. Measures like freshness [18] can also be used for evaluation in case the evaluator has access to the search engine's database and the page as seen by the crawler can be reconstructed.

In order to evaluate stability, we introduce the following measures for  $i > 1$ :

$$forgotten(q, i, s) = |\{\text{TR URLs retrieved by } s \text{ in round } (i - 1) \text{ that exist on the Web and are TR at round } i \text{ but are not retrieved by } s \text{ in round } i\}|.$$

A *dropped URL* is a URL that disappeared from the search engine's database, even though it still exists on the Web and continues to be technically relevant;  $forgotten(q, i, s)$  counts the number of *dropped* URLs in round  $i$ . A URL  $u$  that was dropped in round  $i$  may reappear in the database at some later round  $j$  (our experience shows that this does happen). Such URLs are called *rediscovered URLs*.  $Recovered(q, j, s)$  counts the number of rediscovered URLs by  $s$  in round  $i > 1$ :

$$recovered(q, j) = |\{\text{TR URLs retrieved by } s \text{ in round } j \text{ that were } \textit{dropped} \text{ by } s \text{ in round } i, i < j \text{ AND were not retrieved by } s \text{ in round } j - 1\}|.$$

If a URL  $u$  appeared for the first time in round  $k$ , and was dropped in round  $i > k$ , and reappeared in round  $j > i$ , and was retrieved again in round  $j + 1$ , it will be *rediscovered* in round  $j$ , but not in round  $j + 1$ . That is, a URL is counted in recovered in the first round it reappears after being *dropped*. It may be the case that the URL was *dropped* in round  $i$  because the server on which it resides was down

at the time the crawler tried to visit it. This may account for some (small part) of *forgotten*. In Sect. 5 we discussed several other possible explanations.

One of these explanations relates to the detection and removal of duplicates. It is well known that there is a lot of content duplication on the Web. Some of it is intentional (mirror sites), some duplication results from simple copying of Web pages, and some is due to different aliases of the same physical address. Thus it is conceivable that the search engine dropped a given URL  $u$  because it located another URL  $u'$  in its database with exactly the same content. To evaluate the extent to which content is lost we define:

$$\begin{aligned} lost(q, i, s) = & |\{\text{URLs dropped by } s \text{ in round } i \\ & \text{for which there is no other URL that was retrieved by } s \\ & \text{for } q \text{ in round } i, \text{ with exactly the same content}\}|. \end{aligned}$$

*Lost URLs* are those URLs that were *dropped*, and for which the search engine did not retrieve any content duplicates of these URLs in the current search round. These URLs cause real information loss for the users, since information that was accessible through the search engine before is not accessible anymore, even though the information is still relevant and available on the Web. The results of a case study [7] show that not only were a high percentage of the URLs *dropped* and *rediscovered*, but a significant portion of them were also lost.

A URL can be dropped and then rediscovered several times during the search period. In order to assess the search performance over the whole search period we define:

$$\begin{aligned} well\_handled(q, s) = & |\{\text{TR URLs retrieved by } s \text{ for } q \text{ that were never} \\ & \text{dropped during the search period}\}|. \end{aligned}$$

The URLs counted in *well\_handled* are not necessarily retrieved during the whole search period. Such a URL can first appear in round  $i > 1$  and disappear from the list of retrieved URL in round  $j > i$ , if it disappears from the Web or ceases to be *technically relevant* to the query. A *mishandled* URL is a URL that was dropped at least once during the search period. Recall that *dropped* means that the URL wrongfully disappeared from the list of URLs retrieved for the query.

$$mishandled(q) = |\{\bigcup \text{dropped URLs, for } i > 1\}|.$$

The set of *mishandled* URLs can be further partitioned into *mishandled\_forgotten*( $q$ ), these are the URLs that were not rediscovered at some later time, and to *mishandled\_recovered*( $q$ ). The last two measures assess the variability of the search results over time and supplement the measures *new* and *forgotten*:

$$\begin{aligned} self\_overlap(q, i, j, s) = & |\{\text{TR URLs retrieved in rounds } i \text{ and } j \text{ by } s, i < j\}| / \\ & |\{\text{TR URLs retrieved by } s \text{ in round } j\}|. \end{aligned}$$

Let  $All(q)$  denote the set of all technically relevant URLs that were retrieved for the query  $q$  by the search engine  $s$  during the whole search period. Note that this is



a virtual set, since it may include URLs that never coexisted on the Web at the same time. Then

$$self\_overlap(q, i, s) = |\{\text{TR URLs retrieved by } s \text{ in round } i\}| / |\{All(q)\}|.$$

High self-overlap for all search rounds indicates that the search engine results for the given query are stable. Note that very high values of self-overlap not only indicate stability, but may also be warning signs that the search engine's database is becoming out-of-date and has not changed substantially for a long period of time. Thus for this measure the "optimal" values are neither very high nor very low.

### 6.3 Evaluating Using These Measures

The initial study [5] that brought to our attention search engine instability was already reviewed in Sect. 5. We carried out a second case study [7] evaluating most of the above-defined measures for a whole year during 2000. The query in the case study was "aporocactus". This word has few (if any) morphological variants. The query was run on six search engines (AltaVista, Excite, Fast, Google, HotBot, and Northern Light) in parallel. The search engines *mishandled* between 33 and 89% of the *technically relevant* URLs retrieved by them during the whole search period. Altogether 1694 different *technically relevant* URLs were located during the whole search period. In this study, the search engines that mishandled the largest percentages of URLs were Google (89%) and HotBot (51%), even though Google retrieved by far the largest number of technically relevant URLs during the whole period – Google covered more than 70% of the set *All* (1694 URLs). Rather interestingly, in spite of Google's large coverage, it usually was not the first to locate *new* URLs. An exception to this was August 2000, shortly after Google underwent a major extension of its database. The fine-tuning of working with this extended database may also explain some of the observed fluctuations. Except for Northern Light, almost all of the *forgotten* URLs were also *lost*, i.e., we were unable to locate in the search results another URL with exactly the same content.

Naturally, we cannot draw any definite conclusions about specific search engines based on two queries ("informetrics OR informetric" and "aporocactus") during two search periods, but the case studies indicate the usefulness of the evaluation criteria defined in this paper.

## 7 Conclusions and Open Problems

In this chapter we discussed how search engines do and should operate in a highly dynamic environment, the Web. The search engines face difficulties unparalleled to traditional IR systems. Previous studies show that the search engines have problems handling the situation. Traditional measures to evaluate search engines are not sufficient since they do not address dynamic changes at all. We introduced a set of measures that assess search engine performance in a dynamic environment.

To summarize, the “optimal search engine” should have high values for *totally new* (relative to other engines), indicating that the search engine picks up new URLs relevant to the query quickly. Ideally, the number of *mishandled* URLs should be zero, but as we explained before, the search engine has to decide how to utilize its available resources, and has to compromise between adding new pages and removing old ones. The number of *broken links* should also approach zero, while *self\_overlap* should neither be very high nor very low.

A *lost* URL may have information on the search topic that is not available from any of the other URLs retrieved by the search engine at a given time, even though this information exists on the Web. In this case the search engine does not able the user to access information that was previously available through the search engine. Even *forgotten* but not *lost* URLs have a negative effect on the user’s search experience. The user may recall a previously visited, useful URL when presented with a list of results, however when this URL does not appear in the search results, in most cases she will not be able to locate this page again.

When counting *dropped* and *lost* URLs we may also want to look at the rank of these URLs. Are these mostly low-ranked URLs or is the distribution more or less uniform? Dropping low-ranked URLs would correspond to the policy announced by Inktomi [50] of removing unpopular URLs from the database. The above-mentioned case studies did not look at the ranks of the dropped URLs.

The criteria introduced here are a first step in defining a set of measures for evaluating search engine performance in dynamic environments. Future work should be carried out both in the theoretical and in the practical directions. We need to define additional criteria and to refine existing ones, and we need to carry out additional larger scale experiments to study the usefulness and the applicability of the measures. We should also look into the effects of commercialization (e.g., paid inclusion, pay for ranking) on search engine results over time.

## References

1. About.com—Web search. List of search engines, 2003. <http://websearch.about.com/cs/searchengines/index.htm>.
2. R. Albert, H. Jeong, and A. L. Barabasi. Diameter of the World-Wide Web. *Nature*, 401:130–131, 1999.
3. A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. S. Raghavan. Searching the Web. *ACM Transactions on Internet Technology*, 1(1):2–43, 2001.
4. R.A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, Harlow, 1999.
5. J. Bar-Ilan. Search engine results over time—A case study on search engine stability. *Cybermetrics*, 2/3, 1999. <http://www.cindoc.csic.es/cybermetrics/articles/v2i1p1.html>.
6. J. Bar-Ilan. Evaluating the stability of the search tools Hotbot and Snap: A case study. *Online Information Review*, 24(6):430–450, 2000.
7. J. Bar-Ilan. Methods for measuring search engine performance over time. *Journal of the American Society for Information Science and Technology*, 54(3):308–319, 2002.
8. J. Bar-Ilan and Peritz B. C. The life span of a specific topic on the Web; the case of ‘informetrics’: a quantitative analysis. *Scientometrics*, 46(3):371–382, 1999.

9. A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
10. K. Bharat and A.Z. Broder. A technique for measuring the relative size and overlap of public Web search engines. In *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, 1998. <http://www7.scu.edu.au/programme/fullpapers/1937/com1937.htm>.
11. M. C. Bowman, P. B. Danzig, D. R. D. R. Hardy, U. Manber, and M. F. Schwartz. The Harvest information discovery and access system. *Computer Networks and ISDN Systems*, 28(1-2):119–125, 1995.
12. B. E. Brewington and G. Cybenko. How dynamic is the Web? In *Proceedings of the 9th International World Wide Web Conference*, Amsterdam, 2000. <http://www9.org/w9cdrom/264/264.html>.
13. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 9th International World Wide Web Conference*, Brisbane, Australia, 1998. <http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm>.
14. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomlins, and J. Wiener. Graph structure in the Web. In *Proceedings of the 9th International World Wide Web Conference*, Amsterdam, 2000. <http://www9.org/w9cdrom/160/160.html>.
15. A. Z. Broder. Search: Beyond the keyword interface. Panel at WWW10, May 2001, Hong Kong, 2001.
16. A. Z. Broder. A taxonomy of Web search. *SIGIR Forum*, Fall 2002. <http://www.sigir.org/forum/F2002/broder.pdf>.
17. S. Chakrabarti, B. Dom, R. S. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, J. Kleinberg, and D. Gibson. Hypersearching the Web. *Scientific American*, 280(1):54–60, 1999.
18. J. Cho and H. Garcia-Molina. The evolution of the Web and implications for an incremental crawler. In *Proceedings of the 26th International Conference on Very Large Databases*, pages 200–209, 2000.
19. J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. *SIGMOD RECORD*, 29(2):117–128, 2000.
20. H. Chu and M. Rosenthal. Search engines for the World Wide Web: A comparative study and evaluation methodology. In *ASIS96*, 1996. <http://www.asis.org/annual-96/Electronic-Proceedings/chu.htm>.
21. C. W. Cleverdon. Evaluation of operational information retrieval systems. Part 1: Identification of criteria. College of Aeronautics, Cranfield, England, 1964.
22. C. W. Cleverdon. The significance of the Cranfield tests on index languages. In *Proceedings of SIGIR 1991*, pages 3–12, 1991.
23. M. Courtois, W. M. Baer, and M. Stark. Cool tools for searching the Web. *Online*, pages 14–32, November/December 1995.
24. W. Ding and G. Marchionini. A comparative study of Web search engine performance. In *Proceedings of ASIS96*, pages 136–142, 1996.
25. F. Douglass, A. Feldmann, Krishnamurthy B., and J. Mogul. A live study of the World Wide Web. In *Proceedings of the USENIX Symposium on Networking Technologies and Systems*, pages 8–11, Monterey, CA, 1997.
26. D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of Web pages. In *Proceedings of the 12th International World Wide Web Conference*, pages 669–678, Budapest, 2003. <http://research.microsoft.com/aboutmsr/labs/siliconvalley/pubs/p97-fetterly/fetterly.pdf>.
27. M. Gordon and P. Pathak. Finding information on the World Wide Web: The retrieval effectiveness of search engines. *Information Processing and Management*, 35:141–180, 1999.

28. Harvest. FAQ. <http://harvest.sourceforge.net/harvest/doc/html/FAQ-1.html>.
29. D. Hawking, N. Craswell, P. Bailey, and K. Griffiths. Measuring search engine quality. *Information Retrieval*, 4:35–59, 2001.
30. M. R. Henzinger, R. Motwani, and C. Silverstein. Challenges in Web search engines. *SIGIR Forum*, Fall 2002. <http://www.sigir.org/forum/F2002/henzinger.pdf>.
31. C. Holscher and G. Strube. Web search behavior of internet experts and newbies. In *Proceedings of the 9th International World Wide Web Conference*, Amsterdam, 2000. <http://www9.org/w9cdrom/81/81.html>.
32. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, CA, 1998. Also appeared in: *Journal of the ACM*, 46(5), 604–632, 1999. <http://www.cs.cornell.edu/home/kleinber/auth.ps>.
33. W. Koehler. An analysis of Web page and Web site constancy and permanence. *Journal of the American Society for Information Science*, 50(2):162–180, 1999.
34. W. Koehler. Web page change and persistence – A four year longitudinal study. *Journal of the American Society for Information Science and Technology*, 53(2):162–171, 2002.
35. S. Lawrence and L. Giles. Searching the World Wide Web. *Science*, 280:98–100, 1998.
36. S. Lawrence and L. Giles. Accessibility and distribution of information on the Web. *Nature*, 400:107–110, 1999.
37. H. V. Leighton and J. Srivastava. First 20 precision among World Wide Web search services (search engines). *Journal of the American Society for Information Science*, 50:870–881, 1999.
38. W. Mettrop and P. Nieuwenhuysen. Internet search engines - fluctuations in document accessibility. *Journal of Documentation*, 57(5):623–651, 2001.
39. S. Mizzaro. Relevance: The whole history. *Journal of the American Society for Information Science*, 48(9):810–832, 1997.
40. G. Notess. Altavista inconsistencies. Retrieved from <http://searchengineshowdown.com/features/av/inconsistent.shtml>.
41. OCLC. Web characterization project. <http://wcp.oclc.org/>.
42. K. M. Risvik and R. Michelsen. Search engines and Web dynamics. *Computer Networks*, 39:289–302, 2002.
43. R. Rousseau. Daily time series of common single word searches in Altavista and NorthernLight. *Cybermetrics*, 2/3, 1999. <http://www.cindoc.csis.es/cybermetricc/articles/v2i1p2.html>.
44. T. Saracevic. RELEVANCE: A review of and a framework for thinking on the notion in Information Science. *Journal of the American Society for Information Science*, 26:321–343, 1975.
45. E. Selberg and O. Etzioni. On the instability of Web search engines. In *Proceedings of RIAO2000*, Paris, 2000. <http://citeseer.nj.nec.com/selberg00instability.html>.
46. C. Silverstein, M. Henzinger, H. Marais, and M. Moricz. Analysis of a very large Web search engine query log. *ACM SIGIR Forum*, 33(1), 1999. <http://www.acm.org/sigir/forum/F99/Silverstein.pdf>.
47. A. Singhal and M. Kaszkiel. A case study in Web search using TREC algorithms. In *Proceedings of the 10th International World Wide Web Conference*, pages 708–716, Hong Kong, 2001. <http://www10.org/cdrom/papers/pdf/p317.pdf>.
48. A. Spink, D. Wolfram, M. B. J. Jansen, and T. T. Saracevic. Searching the Web: The public and their queries. *Journal of the American Society for Information Science and Technology*, 52(3):226–234, 2001.
49. D. Sullivan. How search engines work, 2002. In *Searchenginewatch*. <http://searchenginewatch.com/webmasters/work.html>.

50. D. Sullivan. The search engine update, August 2, 2000. In Searchenginewatch. <http://searchenginewatch.com/subscribers/updates/000802su.html>.
51. J. S. Watson. "If you don't have it, you can't find it." A close look at students' perceptions of using technology. *Journal of the American Society for Information Science*, 49(1):1024–1036, 1998.
52. Webmasters.com Forum. How to tell if a Google update has started, 2002. <http://www.webmasterworld.com/forum3/5582.htm>.
53. Wired Cybrarian. The search engine update, 1997. <http://hotwired.lycos.com/cybrarian/reference/search.html>.
54. X. Zhu and S. Gauch. Incorporating quality metrics in centralized/distributed information retrieval on the World Wide Web. In *Proceedings of ACM SIGIR2000, Research and Development in Information Retrieval*, pages 288–295, Athens, 2000.

## **Part III**

---

### **Events and Change on the Web**

---

## Introduction

Part III of the book discusses techniques for handling events and change in Web applications. Many Web applications need to be *reactive*, that is, able to detect the occurrence of particular events or changes in information content, and to react by automatically executing the appropriate application logic.

Chapters 10–13 discuss the detection and handling of changes in the content of Web-based information. XML has emerged as a standard data format for exchanging and storing information on the Web, and event–condition–action (ECA) rules are a natural candidate for providing reactive functionality over XML repositories by allowing automatic reaction to changes in the XML data. Chapters 10 and 11 describe two languages for expressing ECA rules over XML repositories.

Chapter 10, by Bailey et al., begins with an overview of ECA rule technology and how it has been used in conventional database applications, where ECA rules are also known as triggers. The syntax and semantics of the event, condition and action sublanguages of typical ECA rule languages are discussed, including the SQL3 standard for database triggers, and techniques for analysing the run-time behaviour of sets of ECA rules.

The authors go on to discuss new issues raised in designing an ECA rule language for XML repositories, including the granularity of events and actions on XML data and the need to develop new techniques for rule analysis. They present one particular language for XML ECA rules that is based on a fragment of XPath for its event and condition languages, and on a fragment of XQuery for its action language. They specify the language’s execution model and describe a prototype implementation of it. They also describe techniques for determining the triggering and activation relationships between pairs of rules in their language that can be ‘plugged into’ existing frameworks for ECA rule analysis.

Chapter 11, by Bonifati and Paraboschi, describes another language for defining ECA rules on XML data. This language is more complex than that of Chapt. 10 as it allows full XPath in the event parts of rules, and full XQuery in the condition and action parts. However, analysing the behaviour of ECA rules expressed in this more complex language has not been considered, and there is in general a trade-off between

the complexity of an ECA language on the one hand and the ease of analysing rules expressed in it on the other.

Bonifati and Paraboschi describe two alternative semantics for their language, one where rules are triggered immediately when events occur, and the other where rules are triggered at the end of transactions. Their language adopts an execution model whereby each update of an XML document is decomposed into a sequence of smaller updates, and rules are triggered by the execution of these smaller updates. In contrast, the language of Chapt. 10 treats each XML update as atomic, and rules are triggered only after the whole update has been executed. These semantics may produce different effects for the same initial update, and a detailed investigation of their respective advantages and the performance trade-offs involved is an open research question.

Chapter 11 contains a discussion and categorisation of application areas for XML ECA rules along two dimensions: hand-crafted or tool-generated, and providing services that are internal to the repository or external to the repository. The chapter concludes with a discussion of several directions of further research in the areas of rule expressiveness, analysis and optimisation.

One of the areas of application of XML ECA rules identified by Bonifati and Paraboschi in Chapt. 11 is as a general tool for rapid prototyping of Web services. This theme of Web services is picked up in Chapt. 12, where similar reactive functionality to that provided by ECA rules is obtained by embedding calls to Web services within XML documents, in a framework called Active XML (AXML). When a Web service in an AXML document is invoked, the document is expanded with the results returned. Special attributes on AXML nodes are used to control the activation of Web service calls and the lifetime of the returned data. The authors' goal with AXML is to provide a tool that allows flexible peer-to-peer data integration on the Web.

After a review of related work in Web services, data integration, databases, and peer-to-peer computing, Abiteboul et al. describe the syntax and execution semantics of a set of AXML documents and services distributed over a network of peers. They discuss the issue of controlling access to the services provided by a peer and propose two alternative solutions. A prototype implementation of AXML is described, and the authors conclude with a discussion of directions of further research in the areas of design, analysis and optimisation of AXML documents, security protocols, and the possibility of using AXML to develop distributed Web service directories.

Chapters 10–12 assume that changes are internal to a particular Web application and as such their occurrence is detectable relatively easily by the application. The final two chapters in this section are concerned with the open environment of the Web as a whole. Chapter 13 discusses techniques for the detection and notification of changes in HTML or XML documents and describes the architecture of a system called WebVigil that aims to provide this functionality.

Jacob et al. give an overview of existing tools for detecting and notifying changes in Web pages. Unlike previous tools, WebVigil is based on the use of ECA rules. Users register their interest in a particular Web page by means of a 'sentinel'. This allows specification of the type of change that is of interest, the granularity of the change and when the sentinel should be activated and deactivated. Users can either request that



WebVigiL should monitor the page for the specified change at a particular frequency, or that WebVigiL should detect the change as soon as possible after it occurs. Users also have a number of options for specifying when and how they should be notified of the change.

Once a sentinel has been specified, WebVigiL generates a number of ECA rules that it then uses to implement the behaviour specified by the sentinel. Two application areas identified for WebVigiL are distributed software development over the Web and personalised information retrieval from the Web. One of the open research problems identified is detection of changes to dynamic Web pages, i.e. pages generated at run-time by applications.

Chapter 14 discusses more generally the issues involved in event detection and notification in Web applications, extending the discussion beyond the classical Web to sensor networks and ubiquitous and mobile computing. Buchmann et al. begin by analysing the possible patterns of consumer/producer interaction on the Web and note that event-based interaction typically arises in the case where the information producer does not know the consumers of that information. They thus motivate the need for reliable middleware based on the publish/subscribe paradigm that can react to events occurring in heterogeneous distributed environments.

The authors review relevant work in event dissemination and aggregation, decomposition of traditional ECA functionality into a set of separate services, provision of transactional behaviour in distributed heterogeneous environments and mechanisms for distributed event notification. They then describe the architecture of a middleware platform called DREAM that supports this functionality, and four prototypes, each of which implements some part of the overall DREAM functionality. One of these prototypes is a meta-auction broker that mediates between different auction services on the Web. Another is an Internet-enabled car that allows personalised interaction to different situations as defined by a set of ECA rules. Two open problems identified by the authors are the need for a precise definition of the capabilities of a generic event-based platform such as DREAM, and the need for configurability of such platforms to specific application requirements.

---

# An Event–Condition–Action Language for XML

James Bailey<sup>1</sup>, George Papamarkos<sup>2</sup>, Alexandra Poulouvassilis<sup>2</sup>, Peter T. Wood<sup>2</sup>

<sup>1</sup> Department of Computer Science, University of Melbourne,  
Melbourne, Australia  
jbailey@cs.mu.oz.au

<sup>2</sup> School of Computer Science and Information Systems, Birkbeck College, University of  
London, London WC1E 7HX  
{gpapa05, ap, ptw}@dcs.bbk.ac.uk

**Summary.** XML is now a widespread means of exchanging and storing information on the Web. Event–condition–action (ECA) rules are a natural candidate for the support of reactive functionality on XML repositories. This chapter discusses ECA rules in the context of XML data. We give a review of related work, and then define a language for specifying ECA rules on XML repositories. We specify the rule execution model of our language and describe a prototype implementation. We also discuss techniques for analysing the behaviour of sets of ECA rules, in particular for determining the triggering and activation relationships between pairs of rules. We conclude with a discussion of some directions for further research.

## 1 Introduction

XML is becoming a dominant standard for storing and exchanging information on the Web. With its increasing use in applications such as data warehousing, e-commerce and e-learning [1, 17, 18, 24, 28, 30, 35], there is a rapidly growing need for the support of reactive functionality on XML repositories. *Event–condition–action* (ECA) rules are a natural candidate for this.

ECA rules automatically perform actions in response to events provided that stated conditions hold. They are used in conventional data warehouses for incremental maintenance of materialised views, validation and cleansing of data, and maintaining audit trails. By analogy, ECA rules could also be used as an integrating technology for providing this kind of functionality on XML repositories. Further potential uses include checking key and other constraints on XML documents, and performing automatic repairs when violations of constraints are detected. In a ‘push’ type environment, they can be used for automatically broadcasting information to subscribers as the contents of relevant documents change. They can also be employed as a flexible means of maintaining statistics about document and Web site usage and behaviour.

There are two main advantages in using ECA rules to support such functionality as opposed to implementing it directly using a programming language such as Java. First, ECA rules allow an application’s reactive functionality to be defined and

managed within a single rule base rather than being encoded in diverse programs. This enhances the modularity and maintainability of such applications. Second, ECA rules have a high-level, declarative syntax and are thus amenable to powerful analysis and optimisation techniques, which cannot be applied if the same functionality is expressed directly in programming language code.

An alternative way to implement the functionality described above might be to use XSLT to transform source XML documents. However, XSLT would have to process an entire document after any update to it in order to produce a new document, whereas we are concerned with the detection and subsequent processing of updates of much finer granularity. Also, using ECA rules allows direct update of a document, whereas XSLT requires a new result tree to be generated by applying transformations to the source document.

ECA rules have been used in many settings, including active databases [37, 33], workflow management, network management, personalisation and publish/subscribe technology [4, 17, 18, 20, 34], and specifying and implementing business processes [3, 19, 28]. In this chapter, we study ECA rules in the context of XML data. We begin with a review of related work. We then define a language for specifying ECA rules on XML repositories, illustrating the language by means of some examples. We specify the rule execution model of the language and describe a prototype implementation. We also discuss techniques for analysing the behaviour of sets of ECA rules defined in our language, in particular for determining the triggering and activation relationships between pairs of rules. We conclude with a discussion of directions for further research.

## 2 Event–Condition–Action Rules

An ECA rule has the general syntax

*on event if condition do actions*

The event part describes a situation of interest and dictates *when* the rule should be triggered. Events can be broadly classified into two categories: *primitive events* and *composite events*. Primitive events are atomic, detectable occurrences such as database updates (e.g. ‘on insert into relation *R*’) or the reaching of a particular time (e.g. ‘on 13th March 2003 at 15:00’). Composite events are combinations of primitive events, specified using an *event algebra* [22, 26]. Common operators in event algebras include (i) disjunction: event  $e_1 \vee e_2$  occurs if either event  $e_1$  occurs or event  $e_2$  occurs, (ii) sequence: event  $e_1; e_2$  occurs if  $e_2$  occurs, having been preceded by  $e_1$ , and (iii) conjunction: event  $e_1 \wedge e_2$  occurs when both  $e_1$  and  $e_2$  have occurred in any order.

However, most implementations of ECA systems do not support an event algebra as rich as this. Rather, they settle for being able to detect just an appropriate set of primitive events, with no support of event operators. While this limits the range of situations that can be reacted to, rule execution can be more easily optimised and analysed.

Events can have associated with them parameters which provide extra information about the event occurrence. For example, in active databases, these parameters are known as *deltas* and may be referenced by the condition and action parts of the ECA rule. For each event  $E$  detectable by the database there are two deltas: *has\_occurred\_E* and *change\_E*. The former is nonempty if event  $E$  occurred during the execution of the last action, and it contains information about the occurrences of event  $E$ , e.g. the time at which they occurred and which transaction caused them to occur. The delta *change\_E* contains information about the changes that occurrences of event  $E$  made to the database during the execution of the last action.

For example, in a typical active relational database there may be for each user-defined relation  $R$  a set of six delta relations:

- *has\_occurred\_insertion\_R*, which would be nonempty if one or more INSERT statements on relation  $R$  occurred during the execution of the last action;
- *change\_insertion\_R*, which would contain the set of new tuples inserted into relation  $R$  during the execution of the last action;
- *has\_occurred\_deletion\_R*, would be nonempty if one or more DELETE statements on relation  $R$  occurred during the execution of the last action;
- *change\_deletion\_R*, would contain the set of tuples deleted from  $R$  during the execution of the last action;
- *has\_occurred\_update\_R*, would be nonempty if one or more UPDATE statements on relation  $R$  occurred during the execution of the last action;
- *change\_update\_R*, would contain a set of pairs (*old\_tuple*, *new\_tuple*) for each tuple of  $R$  which was updated during the execution of the last action

The event part of an ECA rule is either *has\_occurred\_E* or *change\_E* for some event  $E$ . The identifiers *has\_occurred\_E* and *change\_E* may also occur within the rule's condition and action parts.

A rule is said to be *triggered* if its event part is nonempty. Allowing either *has\_occurred\_E* or *change\_E* to appear as rule events means that both 'syntactic' and 'semantic' triggering of rules can be supported. Syntactic triggering happens if the rule's event part is *has\_occurred\_E* and instances of event  $E$  occur. Semantic triggering happens if the rule's event part is *change\_E* and instances of event  $E$  occur and make changes to the database.

The condition part of an ECA rule determines if the database is in particular state. It is a query over the database and its environment, and its semantics are the same as that used for the database query language, e.g. SQL. The condition may also refer to the state before the execution of the event and the state created after the execution, by making use of the deltas.

The action part of a rule describes the logic to be performed if the condition evaluates to True. It is usually a sequence of modifications applied to the database, expressed using the same syntax as that used by updates within a transaction.

More details on the foundations of ECA rules in active databases and descriptions of a range of implemented active database prototypes can be found in [33, 37].

## 2.1 Rule Execution Model

The rule execution model is a specification of the run-time behaviour of the system. In particular, it specifies when the various components of a rule are executed with respect to one another, and what happens when multiple rules are triggered simultaneously.

This first aspect is traditionally handled by the use of *coupling modes* [21]. A coupling mode specifies the timing activation of one part of an ECA rule with respect to another. Possible coupling modes for the condition part with respect to the event part are:

- Immediate: The condition is evaluated immediately the event is detected as having occurred within the current transaction.
- Deferred: The condition is evaluated within the same transaction, but after the last operation in the transaction and just before the transaction commits.
- Decoupled: The condition is evaluated within a separate, child transaction.

Possible coupling modes for the actions part with respect to the condition part are similar:

- Immediate: The action is executed immediately after the condition has been evaluated (if the condition is found to be True).
- Deferred: The action is performed within the same transaction, but after the last operation in the transaction and just before the transaction commits.
- Decoupled: The action is performed within a separate, child transaction.

Different types of coupling modes may be more or less suitable for certain categories of rules. For example, decoupled execution can help response time since the length of transactions does not grow too large because of rule execution and hence potentially more concurrency is available. Decoupled execution can also be useful in situations where the parent transaction aborts, yet rule execution is nevertheless desired in the child transaction, e.g. updating an access log regardless of whether or not authorisation is granted. Maintaining views is typically done immediately to ensure freshness, and either deferred or immediate coupling can be used for checking integrity constraints (immediate for constraints that should never be violated, and deferred for constraints that need only be satisfied when the database is in a stable state).

The second aspect of rule execution is the policy employed for determining which rule to execute next, given that several rules have been previously triggered and are awaiting execution. The time of triggering is an important factor here and thus maintaining rules in a data structure which reflects this timing information is natural, e.g. a first-in-first-out or a last-in-first-out list. For rules which were triggered at precisely the same time by the same event occurrence, further information such as priorities can be used for tie-breaking; each rule is assigned a unique priority and rules with higher priority are executed earlier.

## 2.2 The SQL3 Standard

The SQL3 standard specifies a syntax and execution model for ECA rules, or *triggers*, in relational databases [31]. Rule event parts may be triggered by update, insert or delete operations on the database. Triggers are of two kinds: BEFORE triggers and AFTER triggers. The former conceptually execute the condition and action before the triggering event is executed. The latter execute the condition and action after the triggering event is executed, using an immediate coupling mode between both event and condition, and between condition and action.

Conditions are evaluated on the database state that the action is executed on, and multiply triggered rules are handled using a last-in-first-out list. Each rule is assigned a unique priority. Only syntactic triggering is supported, i.e. the event parts of triggers have the semantics of the *has\_occurred* deltas we described above.

Another important aspect is that of rule granularity, and two types of granularity are supported, *row-level* and *statement-level*. When a statement-level rule is triggered by some event *E* and is then scheduled for execution, one copy of its action part is placed on the list of pending rules. When a row-level rule is triggered by some event *E*, one copy of its actions part is placed on the pending list for each member of *change\_E* for which the rule's condition evaluates to True. Hence, a single event can give rise to zero, one, or many copies of a triggered rule's actions for row-level rules.

## 2.3 ECA Rules for Object-Oriented Databases

Because of the richness of the object-oriented data model, ECA rules for object-oriented databases often contain additional features compared with ECA rules for relational databases. The principal difference is the availability of a richer set of primitive event types, for example, events which are triggered on the invocation of methods or on the creation of objects. Here again, in the same way as for relational databases, deltas can be used to define event contexts. So in a typical active object-oriented database there may be classes *has\_occurred\_M* and *change\_M* for each method *M* whose invocation is detectable as an event by the database. The delta *has\_occurred\_M* would be nonempty if method *M* was invoked during the execution of the last action. The delta *change\_M* would contain information about changes made to user-defined database objects by invocations of method *M* during the last action. Another important difference in active object-oriented databases stems from the ability to specify rules as objects. Relationships between rules can then be captured, using properties such as generalisation and specialisation between rule classes.

## 2.4 Analysing Rule Behaviour

When multiple ECA rules are defined within a system, their interactions can be difficult to predict, since the execution of one rule may cause an event which triggers another rule or set of rules. These rules may in turn trigger further rules, and there is indeed the potential for an infinite cascade of rule firings to occur.

Analysing the behaviour of ECA rules is a well-studied topic in active databases and a number of analysis techniques have been proposed, e.g. [5, 6, 8, 11, 12, 13, 14, 19, 25, 29]. Two key properties of a set of ECA rules are the *triggering* [5] and *activation* [13] relationships between pairs of rules, since this information can be used to analyse properties such as *termination* of the ECA rule set, or *reachability* of specific rules. The triggering and activation relationships between pairs of rules are defined as follows:

- A rule  $r_i$  *may trigger* a rule  $r_j$  if execution of the action of  $r_i$  may generate an event which triggers  $r_j$ .
- A rule  $r_i$  *may activate* another rule  $r_j$  if  $r_j$ 's condition may be changed from False to True after the execution of  $r_i$ 's action.
- A rule  $r_i$  *may activate* itself if its condition may be True after the execution of its action.

The *triggering graph* [5] represents each rule as a vertex, and there is a directed arc from a vertex  $r_i$  to a vertex  $r_j$  if  $r_i$  *may trigger*  $r_j$ . Acyclicity of the triggering graph implies definite termination of rule execution. Triggering graphs can also be used for deriving rule reachability information. The *activation graph* [13] also represents rules as vertices. In this case there is a directed arc from a vertex  $r_i$  to a vertex  $r_j$  if  $r_i$  *may activate*  $r_j$ . Acyclicity of this graph also implies definite termination of rule execution.

Triggering and activation graphs were combined in [11] in a method called *rule reduction*, which gives more precise results than either of the triggering or activation graphs alone. With this method, any vertex that does not have both an incoming triggering and activation arc can be removed from the graph, along with its outgoing arcs. This removal of vertices is repeated until there are no such vertices. If the procedure results in all the vertices being removed, then the rule set is definitely terminating.

More recently, we proposed using *abstract interpretation* to analyse ECA rules [6, 8]. With this approach, the ECA rules are 'executed' on an abstract database representing a number of real databases. Unlike the graph-based approaches, this technique tracks how the triggering and activation relationships between rules *evolve* during rule execution.

Determining triggering and activation relationships between ECA rules is more complex for semistructured data such as XML than for structured databases, because determining the effects of rule actions is not simply a matter of matching up the names of updated database objects with the event and condition parts of ECA rules. Instead, the associations between actions and events/conditions are more implicit, and more sophisticated semantic comparisons between sets of path expressions are required. In Sect. 4 we discuss techniques for determining the triggering and activation relationships for our XML ECA rules.

## 2.5 ECA Rules for XML

The semistructured nature of XML data gives rise to new issues affecting the use of ECA rules. These issues are principally linked to choice of appropriate language syntax and execution model:

- *Event granularity*: In the relational model, the granularity of data manipulation events is straightforward, since insert, delete or update events occur when a relation is inserted into, deleted from or updated, respectively. With XML, this kind of strong typing of events no longer exists. Specifying the granularity of where data has been inserted or deleted within an XML document becomes more complex and path expressions that identify locations within the document now become necessary.
- *Action granularity*: Again in the relational model, the effect of data manipulation actions is straightforward, since an insert, delete or update action can only affect tuples in a single relation. With XML, actions now manipulate entire subdocuments, and the insertion or deletion of subdocuments can trigger a set of different events. Thus, analysis of which events are triggered by an action can no longer be based on syntax alone. Also, the choice of an appropriate action language for XML is not obvious, since there is as yet no standard for an XML update language.

Compared to rules for relational databases, ECA rules for XML data are more difficult to analyse, due to the richer types of events and actions. However, rules for XML have arguably less analysis complexity than rules for object-oriented data. This stems from the fact that object-oriented databases may permit arbitrary method calls to trigger events, and determining triggering relationships between rules may therefore be as difficult as analysing a program written in a language such as C++ or Java. ECA rules for XML, in contrast, can be based on declarative languages such as XQuery and XPath, and so are more amenable to analysis, particularly with the use of natural syntactic restrictions, as in the language we describe in this chapter.

In recent work [9, 10], we specified a language for defining ECA rules on XML data, based on the XPath and XQuery standards. We also developed techniques for analysing the triggering and activation relationships between such rules. This language and the analysis techniques are the subject of the rest of this chapter. A number of other ECA rule languages for XML have also been proposed, although none of this work has focused on analysing the behaviour of the ECA rules:

Bonifati et al. [17] discuss extending XML repositories with ECA rules in order to support e-services. Active extensions to the XSLT [40] and Lorel [2] languages are proposed that handle insertion, deletion and update events on XML documents. Bonifati et al. [18] discuss a more specific application of the approach to push technology, where rule actions are methods that cannot update the repository, and hence cannot trigger other rules.

Bonifati et al. [16] also define an active rule language for XML, which is discussed in detail in the chapter by Bonifati and Paraboschi in this book. The rule syntax of this language is similar to ours, and is based on the syntax of triggers in SQL3. The rule execution model is rather different from ours though. Generally speaking,



insertions and deletions of XML data may involve document fragments of unbounded size. Bonifati et al. [16] adopt an execution model whereby each top-level update is decomposed into a sequence of smaller updates (depending on the contents of the fragment being inserted/deleted) and then triggering of rules is interleaved with the execution of these smaller updates. In contrast, in our language we treat each top-level update as atomic and rules are triggered only after completion of the top-level update. In general, these semantics may produce different results for the same top-level update, and it is a question of future research to determine their respective performance trade-offs and suitability in different application situations.

ARML [23] provides an XML-based rule description for rule sharing among different heterogeneous ECA rule processing systems. In contrast to our language, conditions and actions are defined abstractly as XML-RPC methods, which are later matched with system-specific methods.

GRML [36] is a multipurpose rule markup language for defining integrity, derivation and ECA rules. GRML uses an abstract syntax based on RuleML, leaving the mapping to a real language for each underlying system implementation. GRML aims to provide semantics for defining access over distributed, heterogeneous data sources for rule evaluation and allows the user to declare most of the semantics necessary for processing a rule, and to evaluate events and conditions coming from heterogeneous data sources.

Other related work is [35], which proposes extensions to the XQuery language [41] to incorporate update operations. We refer the reader to that paper for a review of the provision of update facilities in other XML manipulation languages. The update operations proposed are more expressive than the actions supported by our ECA rule language since they also include renaming and replacement operations, and specification of updates at multiple levels of documents. Triggers are discussed in [35] as an implementation mechanism for deletion operations on the underlying relational store of the XML. However, provision of ECA rules at the ‘logical’ XML level is not considered.

### 3 Our ECA Rule Language for XML

An XML repository consists of a set of XML documents. In our language, ECA rules on XML repositories take the following form:

*on event if condition do actions*

We use the XPath [39] and XQuery [41] languages to specify the event, condition and actions parts of rules. XPath is used for selecting and matching fragments of XML documents within the event and condition parts. XQuery is used within insertion actions, where there is a need to be able to construct new XML fragments.

The *event* part of an ECA rule is an expression of the form

INSERT *e*

or

DELETE *e*

where *e* is a *simple XPath expression* (see Sect. 3.1) which evaluates to a set of nodes.

The rule is *triggered* if this set of nodes includes any node in a new XML fragment, in the case of an insertion, or in a deleted fragment, in the case of a deletion. The system-defined variable  $\$delta$  is available for use within the condition and actions parts of the rule, and its set of instantiations is the set of new or deleted nodes returned by  $e$ . The *condition* part of a rule is either the constant TRUE, or one or more simple XPath expressions connected by the boolean connectives and, or, not. The *actions* part of a rule is a sequence of one or more actions:

$$action_1; \dots; action_n$$

where each  $action_i$  is an expression of one of the following three forms:

INSERT  $r$  BELOW  $e$  BEFORE  $q$

INSERT  $r$  BELOW  $e$  AFTER  $q$

DELETE  $e$

Here,  $r$  is a *simple XQuery expression*,  $e$  is a *simple XPath expression* and  $q$  is either the constant TRUE or an *XPath qualifier* (see Sect. 3.1 for definitions of the italicised terms).

In an INSERT action, the expression  $e$  specifies the set of nodes  $N$  immediately below which new XML fragment(s) will be inserted. These fragments are specified by the expression  $r$ . If  $e$  or  $r$  references the  $\$delta$  variable, then one XML fragment is constructed for each instantiation of  $\$delta$  for which the rule's condition evaluates to True. If neither  $e$  nor  $r$  references  $\$delta$ , then a single fragment is constructed. The expression  $q$  is an XPath qualifier that is evaluated on each child of each node  $n \in N$ . For insertions of the form AFTER  $q$ , the new fragment(s) are inserted after the last sibling for which  $q$  is True, while for insertions of the form BEFORE  $q$ , the new fragment(s) are inserted before the first sibling for which  $q$  is True. The order in which new fragments are inserted is nondeterministic.

In a DELETE action, the expression  $e$  specifies the set of nodes which will be deleted (together with their descendant nodes). Again,  $e$  may reference the  $\$delta$  variable.

*Example 1.* Consider an XML repository containing an XML document  $s.xml$  that contains information about share prices on a stock exchange. We show below some of the information held for a particular share in the document, share XYZ. Under day-info the share price is recorded periodically for the specified date. The highest and lowest prices for each day and each month are also recorded, under day-info and month-info, respectively.

```
<shares>
...
<share name="XYZ">
...
  <day-info day="03" month="03">
    <prices>
      <price time="09:00">123.25</price>
      <price time="09:05">123.50</price>
      <price time="09:10">123.00</price>
    </prices>
```

```

    <high>123.50</high>
    <low>123.00</low>
  </day-info>
  ...
  <month-info month="03">
    <high>133.75</high>
    <low>111.25</low>
  </month-info>
</share>
<share name="ABC">
  ...
</share>
...
</shares>

```

Suppose that this document is updated in response to external events received from a share price information service. In particular, an insertion event will arrive periodically with the current price for share XYZ. For example, such an insertion event *Ev* might be the following update, which inserts the new share price of 123.75 after the last share price currently recorded:

```

INSERT <price time="09:15">123.75</price>
      BELOW document('s.xml')/shares/share[@name="XYZ"] /
          day-info[@day="03"][@month="03"]/prices
      AFTER TRUE

```

The following ECA rule  $r_1$  checks whether the daily high needs to be updated in response to a new price insertion in some share:

```

on INSERT document('s.xml')/shares/share/day-info/
    prices/price
if $delta > $delta/../../high
do DELETE $delta/../../high;
    INSERT <high>$delta/text()</high>
    BELOW $delta/../../ AFTER prices

```

Here, the  $\$delta$  variable is bound to the newly inserted price node detected by the event part of the rule. The rule's condition checks that the value of this price node is greater than the value of the high node under the same day-info node. The action then deletes the existing high node and inserts a high node whose value is that of the newly inserted price. The insertion event *Ev* above would trigger this rule  $r_1$ , which would then update the daily high of share XYZ to 123.75.

The following ECA rule,  $r_2$ , similarly checks whether the monthly high price for a share needs to be updated in response to an insertion of a new daily high price:

```

on INSERT document('s.xml')/shares/share/day-info/
    high

```

```

if $delta > $delta/../../month-info[@month=$delta/../../
    @month]/high
do DELETE $delta/../../month-info/high;
   INSERT $delta
        BELOW $delta/../../month-info
            [@month=$delta/../../@month]
        BEFORE TRUE

```

In the INSERT action of this rule, a copy of the `high` node whose insertion triggered the rule is inserted as the first child of the corresponding `month-info` node.

The insertion event  $Ev$  above would trigger rule  $r_1$ , and the second action of  $r_1$  would in turn trigger rule  $r_2$ . However, the condition of  $r_2$  would then evaluate to False and so its action would not be executed. Similar ECA rules could be used to update the daily and monthly low prices, and for undertaking many other potentially useful tasks.

### 3.1 Simple XPath and XQuery Expressions

The XPath and XQuery expressions appearing in our ECA rules are restrictions of the full XPath and XQuery languages, to what we term *simple* XPath and XQuery expressions. These represent useful and reasonably expressive fragments which have the advantage of also being amenable to analysis, a topic we discuss in Sect. 4.

The XPath fragment we use disallows a number of features of the full XPath language, most notably the use of any axis other than the child, parent, self or descendant-or-self axes and the use of all functions other than `document()` and `text()`. Thus, the syntax of a *simple XPath expression*  $e$  is given by the following grammar, where  $s$  denotes a string and  $n$  denotes an element or attribute name:

$$\begin{aligned}
 e &::= \text{'document(' } s \text{' } (( \text{'/' } \mid \text{'//'} ) p) \mid \\
 &\quad \text{'$delta' } ( \text{'[' } q \text{' ]'} )^* (( \text{'/' } \mid \text{'//'} ) p)? \\
 p &::= p \text{'/' } p \mid p \text{'//'} p \mid p \text{'[' } q \text{' ]'} \mid n \mid \text{'*'} \mid \\
 &\quad \text{'@'} n \mid \text{'@*'} \mid \text{'.' } \mid \text{'..'} \mid \text{'text(' } \text{' } \text{' )'} \\
 q &::= q \text{'and'} q \mid q \text{'or'} q \mid e \mid p \mid (p \mid e \mid s) o (p \mid e \mid s) \\
 o &::= \text{'=' } \mid \text{'!='} \mid \text{'<=' } \mid \text{'<'} \mid \text{'>=' } \mid \text{'>'}
 \end{aligned}$$

Expressions enclosed in `'[` and `']` in an XPath expression are called *qualifiers*. So a simple XPath expression starts by establishing a context, either by a call to the `document` function followed by a path expression  $p$ , or by a reference to the variable `$delta` (the only variable allowed) followed by optional qualifiers  $q$  and an optional path expression  $p$ . Note that a qualifier  $q$  can comprise a simple XPath expression  $e$ .

The XQuery fragment we adopt disallows the use of full FLWR expressions (involving the keywords `'for'`, `'let'`, `'where'` and `'return'`), essentially permitting only the `'return'` part of such an expression [41]. The syntax of a *simple XQuery expression*  $r$  is given by the following grammar:

$$\begin{aligned}
r &::= e \mid c \\
c &::= \langle n \ a \ (\langle / \rangle \mid (\langle \rangle \ t \ast \langle / \rangle \ n \ \langle \rangle)) \\
a &::= (n \ \langle = \ \rangle \ (s \mid e') \ \langle \rangle \ a)? \\
t &::= s \mid c \mid e' \\
e' &::= \{ e \}
\end{aligned}$$

Thus, an XQuery expression  $r$  is either a simple XPath expression  $e$  (as defined above) or an element constructor  $c$ . An element constructor is either an empty element or an element with a sequence of element contents  $t$ . In each case, the element can have a list of attributes  $a$ . An attribute list  $a$  can be empty or is a name equated to an attribute value followed by an attribute list. An attribute value is either a string  $s$  or an *enclosed expression*  $e'$ . Element contents  $t$  is one of a string, an element constructor or an enclosed expression. An enclosed expression  $e'$  is an XPath expression  $e$  enclosed in braces. The braces indicate that  $e$  should be evaluated and the result inserted at the position of  $e$  in the element constructor or attribute value.

### 3.2 Rule Execution Model

We now describe the rule execution model of our language. The input to the execution is a *schedule*  $s$  and an *XML repository*  $db$ . The schedule consists of a list of pairs  $(action_{i,j}, delta_i)$ , where  $action_{i,j}$  is the  $j$ th action within the actions part of rule  $r_i$  and  $delta_i$  is a set of instantiations for the  $\$delta$  variable of rule  $r_i$  for which  $r_i$ 's condition evaluated to True. Rules whose event parts reference the same XML document can potentially be triggered by the same update event on that document. To disambiguate the effect of such rules, we require that all rules whose event parts are insertions on the same document are totally ordered, as are all rules whose event parts are deletions on the same document. The relative priorities of such rules are specified by the user when defining a new rule.

The schedule which initiates rule execution consists of an action and a set of instantiations for the  $\$delta$  variable upon which this action is to be applied, i.e. the initial schedule is a singleton of the form  $[(action, delta)]$ . The following pseudocode expresses how this update request is handled:

```

while s != [] do {
  (a,delta)      := head (s);
  s              := tail (s);
  (changes,db)   := updateDB (db,a,delta);
  for each rule r_i in order of increasing priority {
    if changes[i] != {} then {
      (value,delta) := evalCondition(i,changes[i],db);
      if value = True then
        for j := noOfActions[i] downto 1
          s := (action[i,j],delta):s
    }
  }
}

```

In the above pseudocode, the function `head` returns the first element of a list and the function `tail` returns a list minus its first element.

The function `updateDB` executes the action `a` that was at the head of the schedule. If `a` does not reference the `$delta` variable, this update is performed just once on the repository `db`. If `a` does reference the `$delta` variable, a *set* of updates is generated by substituting occurrences of `$delta` within `a` by each member of `delta`. Thus if  $n$  is the cardinality of `delta`,  $n$  updates will be generated.<sup>3</sup> These updates are then performed in an arbitrary order on the repository.

`updateDB` returns a pair `(changes, db)`, where `db` is the new repository resulting from the update and `changes` is an array such that `changes[i]` is the set of newly inserted or newly deleted nodes corresponding to the event part of rule `r.i`. In particular, if `a` is an insertion then for each `r.i` which may be triggered by `a`, the event part of `r.i` is evaluated on the repository after `a` is executed, and `changes[i]` is the intersection of this result and the new nodes inserted by `a`. If `a` is a deletion then for each `r.i` which may be triggered by `a`, the event part of `r.i` is evaluated on the repository before `a` is executed, and `changes[i]` is the intersection of this result and the nodes that are subsequently deleted by `a`.<sup>4</sup>

The function `evalCondition` evaluates rule `r.i`'s condition, and there are two possible cases:

1. If the `$delta` variable occurs in the condition, then the condition is evaluated once for each member of `changes[i]`, and the subset of `changes[i]` for which it evaluates to `True` is determined. The variable `delta` is set to this subset. If `delta` is nonempty, then the variable `value` is set to `True`; otherwise it is set to `False`.
2. If the `$delta` variable does not occur in the condition, then the condition is evaluated just once and the variable `value` is set to the result. The variable `delta` is set to `changes[i]`.

`noOfActions[i]` is the number of actions in the actions part of rule `r.i` and `actions[i, j]` is the  $j$ th action of `r.i`. The loop `for j:=noOfActions[i] downto 1 do ...` ensures that the actions of a given rule are placed in the right order onto the schedule. Each such action `action[i, j]` is paired with that rule's `delta` and prefixed to the current schedule by the statement `s := (action[i, j], delta):s`.

Rules are considered in increasing order of their priority in the outer `for` loop. Thus the actions of higher-priority rules that have fired will be placed onto the schedule in front of the actions of lower-priority rules.

<sup>3</sup>  $n$  is guaranteed to be finite due to the syntax of our update language, which does not allow infinite new XML fragments to be created, and the fact that there are a finite number of ECA rules.

<sup>4</sup> We will see in Sect. 4 how the set of rules that may be triggered by an action can be determined. It would also be correct to evaluate the event parts of all rules since for those that cannot be triggered by the action, `changes[i]` will necessarily be empty. Thus, limiting the evaluation to the set of rules that may be triggered is an optimisation.

The execution proceeds in this manner until the schedule becomes empty. Nontermination of rule execution is a possibility and thus development of static rule analysis techniques is important to aid the design of ‘well-behaved’ rules. We discuss such techniques in Sect. 4.

There are a number of observations we can make regarding the above rule execution model:

- Triggering is semantic, not syntactic, since a rule  $r_i$  is triggered only if  $\text{changes}[i]$  is not empty.
- The event/condition coupling mode and the condition/action coupling modes are both *Immediate*, since conditions are evaluated immediately after an event becomes true, and the actions of rules that have fired as a result of the current set of updates are placed at the head of the schedule.
- Rule conditions are evaluated against the repository state in which the rule was triggered (unlike in SQL3, where conditions are evaluated against the database state that the action will be executed on). It is possible to evaluate conditions against the database state that the action will be executed on by adding the conditions as additional qualifiers to the XPath expression  $e$  that is part of *INSERT* and *DELETE* actions.
- Both *document-level* and *instance-level* triggering are supported in our ECA rule language, depending on the occurrence of  $\$delta$  in the condition and action parts of a rule:
  - If there is no occurrence of  $\$delta$  in the condition or the action, the action is executed once if the condition is True – this is document-level triggering.
  - If  $\$delta$  occurs in the action (and possibly in the condition), the action is executed once for each possible instantiation of  $\$delta$  for which the condition is True – this is instance-level triggering.

### 3.3 A Prototype Implementation

As a proof of concept, we have developed a prototype system that implements our language and the execution model described above. For this first prototype we have used flat files and have exploited the functionality provided by the W3C DOM standard [43] for interacting with them. The architecture of our system is illustrated in Fig. 1.

The *Parser* parses and checks the syntactic validity of a new rule. For construction of the parser, we have used the JavaCC lexer-parser generator.<sup>5</sup> Valid rules are translated into an XML form and are added by the *Registration Unit* to the *Rule Base* (which is an XML file). Details about each rule are stored here, including its name, priority, and event, condition and action parts.

The *Execution Engine* encapsulates the rule processing functionality. In particular, the *Event Dispatcher*, *Condition Evaluator* and *Action Scheduler* implement these aspects of the rule processing, as we describe in more detail below. All of these

<sup>5</sup> <http://javacc.dev.java.net/>

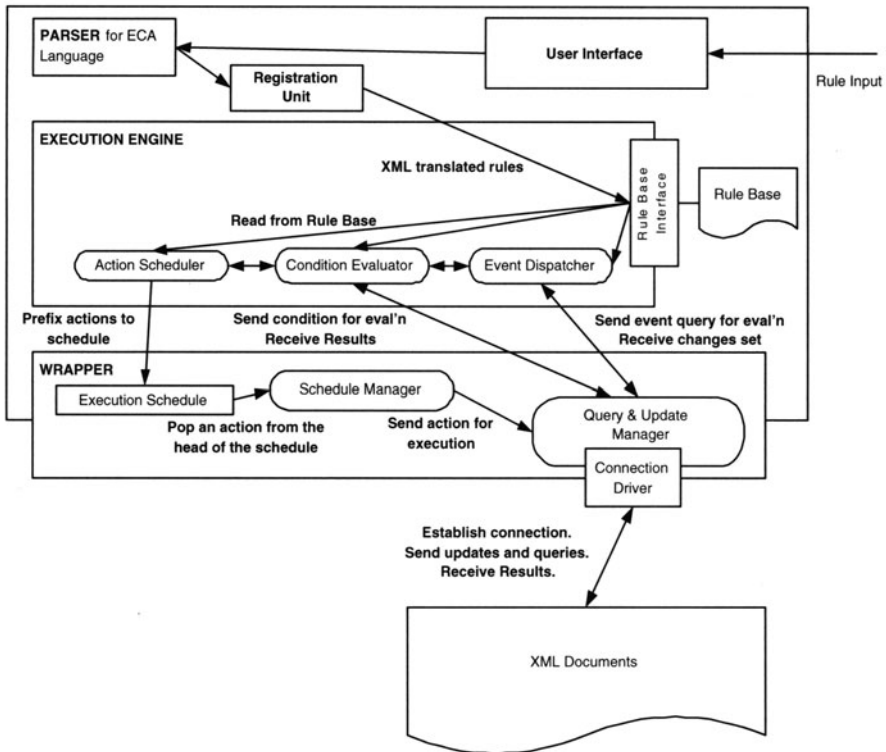


Fig. 1. System architecture

components interface with the *Wrapper* in order to send and receive data to and from the underlying XML files.

The *Execution Schedule* contains a sequence of *updates*, which have the same syntax as rule actions except that they do not contain any *\$delta* expressions within them. By '*\$delta* expression' we mean an XPath expression (either stand-alone or possibly nested within another XPath expression) that starts with *\$delta*. These portions of a rule's action part are replaced by the result of evaluating them on the current document – see below.

The *Wrapper* interfaces with the XML files on disk. All update and query requests from the upper levels of the system pass through this component, which coordinates them. It undertakes to open files, submit queries and updates, and receive back results from them. The *Wrapper* performs these services by using the functionality of the Apache Xalan API. All queries are performed directly by using XPath. For deletions, we identify the set of nodes that will be deleted by using the XPath expression within the *DELETE* part of the request, and we then remove all the subdocuments rooted at the nodes identified. For insertions, we identify the set of nodes that will be affected by using the XPath expression within the *BELOW* part of the request, and we then add the fragment specified within the *INSERT* part as a new child of each of the



nodes identified, placing it relative to the existing children according to the AFTER or BEFORE qualifier.

Rule execution begins with a request from the *Schedule Manager* to the *Query & Update Manager* to execute the update currently at the head of the schedule. In case of an insertion, the Query & Update Manager executes the update and annotates the newly inserted nodes, while in the case of a deletion it annotates the nodes to be deleted without executing the deletion yet.<sup>6</sup>

Following the execution of the update, control then passes to the *Event Dispatcher*. This requests the Query & Update Manager to evaluate the XPath query of the event part of each rule that may be triggered by the update that was just executed. For each rule  $r_i$  whose query result set contains annotated nodes (either newly inserted or about to be deleted), the Event Dispatcher creates the `changes[i]` set containing these annotated nodes, and the rule is triggered.

The *Condition Evaluator* then requests the Query & Update Manager to evaluate the condition part of each triggered rule on the affected document, using as the evaluation context either the root node if there are no occurrences of `$delta` within a query, or each instance of the `changes` set otherwise. The rule's `delta` set is thus created (which will be a subset of its `changes` set). If this is nonempty, the rule fires and control is passed to the Action Scheduler to further process the rule. Otherwise, processing of the rule ends here.

The *Action Scheduler* reformulates a given rule's action(s) in order to eliminate any instances of `$delta` expressions within them. The reformulation algorithm performs the following steps for each node within the rule's `delta` set:

- replaces the `$delta` variable in each of the `$delta` expressions by the current node of the `delta` set;
- evaluates each of the modified `$delta` expressions with respect to the updated document;
- replaces each `$delta` expression within the rule's action(s) by the corresponding result of the previous step

The outcome of this reformulation is that one instance of the rule's action(s) is created for each node in the rule's `delta` set. These updates are now prefixed, in an arbitrary order, to the front of the schedule. Once this has been done for all the rules that have fired, control passes once more to the Schedule Manager and the cycle repeats.

If the last update executed by the Query & Update Manager was a DELETE, then before control passes back to the Schedule Manager, the actual deletion of the annotated nodes is first performed.

*Example 2.* Consider the XML file in Example 1 and the insertion event *Ev*. Initially the schedule consists of just this INSERT update. The Schedule Manager pops this

<sup>6</sup> The annotation of nodes is performed using non-DOM methods provided by Apache Xerces API that allow us to attach data to XML nodes without affecting the physical representation of the file.

update from the schedule and sends it to the Query & Update Manager. This executes the update and annotates the newly inserted nodes. The Event Dispatcher then evaluates the XPath expression in rule  $r_1$ 's event part, which is the only one of the two rules in Example 1 that may be triggered by the update. It finds that the result set contains the newly inserted `price` node and it creates the `changes[1]` set, with this node as its only member. The Condition Evaluator then evaluates the condition part of rule  $r_1$ , finds it to be True for the single element of `changes[1]` and so creates the `delta` set, with the new `price` node as its only member, and passes control to the Action Scheduler.

The Action Scheduler extracts the `$delta` expression `$delta/../../month-info/high` from rule  $r_1$ 's DELETE action and the `$delta` expressions `$delta/text()` and `$delta/../../` from its INSERT action and replaces `$delta` by the one member of the `delta` set (the new `price` node). It submits these three queries to the Wrapper, and then substitutes the resulting values for the three `$delta` expressions within  $r_1$ 's actions. The resulting updates are placed at the front of the schedule and control is passed to the Schedule Manager.

The Schedule Manager pops the first DELETE update from the schedule and sends it to the Query & Update Manager. This 'executes' the deletion by annotating the nodes to be deleted. The Event Dispatcher determines that no rules can be triggered by this update, and the nodes annotated as 'to be deleted' are then actually deleted. The Schedule Manager then pops the next INSERT update from the schedule and sends it to the Query & Update Manager. This executes the update and annotates the newly inserted nodes. The Event Dispatcher evaluates the XPath expression for rule  $r_2$  and finds that the result set contains the newly inserted `high` node. The Condition Evaluator then evaluates the condition part of the rule  $r_2$ , finds it to be False for the single element of `changes[2]` and so no further rules are scheduled. The schedule is now empty, so rule execution terminates.

## 4 Analysing and Optimising Rule Behaviour

Techniques for determining triggering and activation relationships between rules can be utilised in a variety of ways for analysing and optimising the behaviour of XML ECA rules defined in our language:

- They can then be 'plugged into' existing frameworks for ECA rule analysis (both static and dynamic) such as the approaches we reviewed in Sect. 2.4. For example, if we know the pairwise triggering and activation relationships between rules, we can use triggering graph analysis, activation graph analysis or the rule reduction method. It is also possible to use triggering and activation information within an abstract interpretation framework for a more precise analysis than these graph-based approaches, as discussed in [7].
- Information about which ECA rules may be triggered by the current rule action can be used during rule execution to limit the set of rule event parts that need to be evaluated after the execution of this action.

- The activation relationships between pairs of rules can be dynamically updated during rule execution, and this information can be used to avoid evaluating rule conditions which can currently be inferred to be definitely True or False.

#### 4.1 Determining Triggering Relationships

In order to determine triggering relationships between our XML ECA rules, we need to be able to determine whether an action of some rule may trigger the event part of some other rule. Clearly, INSERT actions can only trigger INSERT events, and DELETE actions can only trigger DELETE events.

For any insertion action  $a$  of the form

INSERT  $r$  BELOW  $e_1$  BEFORE|AFTER  $q$

in some rule  $r_i$  and any insertion event  $ev$  of the form

INSERT  $e_2$

in some rule  $r_j$ , we need to know whether event  $ev$  is *independent* of action  $a$ , that is,  $e_2$  can never return any of the nodes inserted by  $a$ .

The XQuery  $r$  defines which nodes are inserted by  $a$ , while the XPath expression  $e_1$  defines where these nodes are inserted. So if it is possible that some initial part of  $e_2$  can specify the same path through some document as  $e_1$  and the remainder of  $e_2$  ‘matches’  $r$ , then  $ev$  is not independent of  $a$ . We now define these notions more formally:

We define a *prefix* of a simple XPath expression  $e$  to be an expression  $e'$  such that  $e = e' / e''$  or  $e = e' // e''$ . We call  $e''$  the *suffix* of  $e$  and  $e'$ . For an XQuery  $r$ , let  $type(r)$  be the result type of  $r$  – this can be determined using the type inference techniques described in [27] or [42]. Using the same techniques, we can test whether or not an XPath expression  $e$  can return a nonempty result when evaluated on documents of  $type(r)$  by first inferring the output type of  $e$ , given input  $type(r)$ , and then checking whether the output type is the inconsistent type. If so, then  $e$  always returns an empty result on input of  $type(r)$ , in which case we say that  $type(r)$  *cannot satisfy*  $e$ . If the output type is not the inconsistent type, we say that  $type(r)$  *may satisfy*  $e$ .

Given XPath expressions  $e_1$  and  $e_2$ , we say that  $e_1$  and  $e_2$  are *independent* if, for all possible XML documents  $d$ ,  $e_1(d) \cap e_2(d) = \emptyset$ . We discuss in [9] how testing for independence of two simple XPath expressions can be done by finding an XPath expression that corresponds to  $e_1 \cap e_2$ , and then checking that the containment  $e_1 \cap e_2 \subseteq \emptyset$  holds (more general fragments of XPath which are also closed under intersection are presented in [15], while the complexity of the containment problem for various fragments of XPath is discussed in [32]).

Thus, event  $ev$  above is independent of action  $a$  if for all prefixes  $e'_2$  of  $e_2$ , either

1.  $e_1$  and  $e'_2$  are independent, or
2.  $type(r)$  cannot satisfy  $e''_2$ .

Equivalently, a rule  $r_i$  (containing action  $a$ ) may trigger rule  $r_j$  (containing event  $ev$ ) if for some prefix  $e'_2$  of  $e_2$ ,  $e_1$  and  $e'_2$  are not independent and  $type(r)$  may satisfy  $e''_2$ .

Similarly to insertions, for any deletion action  $a$  of the form

DELETE  $e_1$

belonging to a rule  $r_i$ , and any deletion event  $ev$  of the form

DELETE  $e_2$

belonging to a rule  $r_j$ , we have that  $r_i$  may trigger  $r_j$  if  $ev$  is not independent of  $a$ . The test for independence of an action and an event in the case of deletions is simpler than for the insertion case above. Let  $e$  be the XPath expression  $e_1 // *$ . Then event  $ev$  is independent of action  $a$  if expressions  $e$  and  $e_2$  are independent.

*Example 3.* Consider the insertion event  $Ev$  and ECA rules  $r_1$  and  $r_2$  from Example 1. We can detect that  $Ev$  may trigger  $r_1$  since (1) this prefix  $e'_2$  of the XPath expression  $e_2$  in the event part of  $r_1$ :

document('s.xml')/shares/share/day-info/prices

and the XPath expression from  $Ev$ :

document('s.xml')/shares/share[@name="XYZ"]/  
day-info[@day="03"][@month="03"]/prices

are not independent, and (2) the type of the XQuery fragment in  $Ev$ , namely `price`, satisfies the suffix  $e''_2$  of  $e_2$  (also `price`). We can also detect that  $Ev$  cannot trigger rule  $r_2$  since every prefix of `document('s.xml')/shares/share/day-info/high` is independent of  $e_1$ .

## 4.2 Determining Activation Relationships

In order to determine activation relationships between our ECA rules, we need to be able to determine

- a. whether an action of some rule  $r_i$  may change the value of the condition part of some other rule  $r_j$  from False to True, in which case  $r_i$  may activate  $r_j$ ; and
- b. whether all the actions of a rule  $r_i$  will definitely leave the condition part of  $r_i$  False, i.e. whether rule  $r_i$  is *self-disactivating*; if not, then  $r_i$  may activate itself

Without loss of generality, we can assume that rule conditions are in disjunctive normal form, i.e. they are of the form

$(l_{1,1} \text{ and } l_{1,2} \dots \text{ and } l_{1,n_1}) \text{ or } (l_{2,1} \text{ and } l_{2,2} \dots \text{ and } l_{2,n_2}) \text{ or } \dots \text{ or } (l_{m,1} \text{ and } l_{m,2} \dots \text{ and } l_{m,n_m})$

where each  $l_{i,j}$  is either a simple XPath expression  $c$ , or the negation of a simple XPath expression, `not c`.

**Simple XPath expressions.** The following table shows the transitions that the truth value of a condition consisting of a single simple XPath expression can undergo. The first column shows the condition's truth value before the update, and the subsequent columns its truth value after a nonindependent insertion (NI) and a nonindependent deletion (ND):

Before	After NI	After ND
<i>True</i>	<i>True</i>	<i>True or False</i>
<i>False</i>	<i>True or False</i>	<i>False</i>

For case (a) above, i.e. when  $r_i$  and  $r_j$  are distinct rules, it is clear from this table that  $r_i$  can activate  $r_j$  only if one of the actions of  $r_i$  is an insertion that is nonindependent of the condition of  $r_j$ .

Let the condition of  $r_j$  be the simple XPath expression  $c$ . The procedure for determining nonindependence of an insertion from a condition  $c$  involves constructing from  $c$  a set  $C$  of conditions, each of which is an XPath expression without any qualifiers. The objective is that condition  $c$  can change from False to True as a result of an insertion only if at least one of the conditions in  $C$  can change from False to True as a result of the insertion. We start with set  $C = \{c\}$  and proceed to decompose  $c$  into a number of conditions without qualifiers, adding each one to  $C$ . See [9] for details of the decomposition algorithm.

Now let one of the actions  $a$  from rule  $r_i$  be

INSERT  $r$  BELOW  $e_1$  BEFORE|AFTER  $q$

We determine  $type(r)$  and consider prefixes and suffixes of each condition  $c_i \in C$ , where  $c_i = c'_i \cdot c''_i$ . Set  $C$  of conditions is independent of  $a$  if for each  $c_i \in C$  and for each prefix  $c'$  of  $c$ , either

1.  $e_1$  and  $c'_i$  are independent, or
2.  $type(r)$  cannot satisfy  $c''_i$

If so, then action  $a$  cannot change the truth value of condition  $c$  in rule  $r_j$  from False to True. Equivalently, we can say that rule  $r_i$  may activate rule  $r_j$  if for some prefix  $c'_i$  of some  $c_i \in C$ ,  $e_1$  and  $c'_i$  are not independent and  $type(r)$  may satisfy  $c''_i$ .

For case (b) above, if the condition part of  $r_i$  is a simple XPath expression  $c$ , the rule will be self-disactivating if all its actions are deletions which subsume  $c$ . For each deletion action

DELETE  $e_1$

we thus need to test if

$$e_1// * \supseteq c$$

For simple XPath expressions and provided additionally that the only operator appearing in qualifiers is '=', it is known that containment is decidable [32]. The decidability of containment for various larger fragments of XPath is shown in [15, 32]. However, even if a fragment of XPath is used for which this property is undecidable, it is still possible to use conservative approximations. For example, if there are occurrences of comparison operators other than '=' in the condition part of a rule, then we can analyse each operand separately against each deletion action and if either operand is subsumed by the action, then we can infer that this action makes this condition False.

*Example 4.* Consider rule  $r_1$  from Example 1. We can detect that its first (DELETE) action makes False its condition, since it deletes the existing *high price*. However, we cannot conclude that the rule is self-disactivating since the second action of  $r_1$  is an INSERT and there is the possibility that this may insert nodes which cause  $r_1$ 's

condition to remain True. Only a deeper analysis of the rule set would detect that rule  $r_1$  is in fact self-disactivating.

**Negations of Simple XPath expressions.** The following table shows the transitions that the truth value of a condition of the form  $\text{not } c$ , where  $c$  is a simple XPath expression, can undergo. The first column shows the truth value of the condition before the update, and the subsequent columns its truth value after a nonindependent insertion (NI) and a nonindependent deletion (ND):

Before	After NI	After ND
<i>True</i>	<i>True or False</i>	<i>True</i>
<i>False</i>	<i>False</i>	<i>True or False</i>

For case (a), where rules  $r_i$  and  $r_j$  are distinct, it is clear from this table that  $r_i$  can activate  $r_j$  only if one of the actions of  $r_i$  is a deletion which is nonindependent of the condition of  $r_j$ .

Let the condition of  $r_j$  be  $\text{not } c$ . We construct the set of conditions  $C$  from  $c$  as outlined in Sect. 4.1. Now let an action from rule  $r_i$  be

DELETE  $e_1$

and let  $e$  be the query  $e_1//*$ . We again use the technique outlined in Sect. 4.1 in order to check whether  $e$  is independent of each of the conditions in  $C$ . If so, then  $e$  cannot change the truth value of  $\text{not } c$  from False to True. Otherwise,  $e$  is deemed to be nonindependent of  $\text{not } c$ , and  $r_i$  may activate  $r_j$ .

For case (b) above, a rule  $r_i$  activates itself if it may leave its own condition True. We again need the notion of a self-disactivating rule. If the condition part of  $r_i$  is  $\text{not } c$ , the rule will be self-disactivating if all its actions are insertions which guarantee that  $c$  will be True after the insertion.

Let an insertion action  $a$  from rule  $r_i$  be

INSERT  $r$  BELOW  $e_1$  BEFORE|AFTER  $q$

and let condition  $c$  comprise prefix  $c'$  and suffix  $c''$ . Action  $a$  guarantees that  $c$  will be True after the insertion if

$$c' \supseteq e_1$$

and *each* of the trees in the set of trees denoted by  $\text{type}(r)$  satisfies  $c''$ . Consequently, we need a stronger concept than the fact that  $\text{type}(r)$  *may satisfy* expression  $c''$ . As in Section 4.1, we can infer the output type of  $c''$ , given input type  $\text{type}(r)$ ; if this output type is equivalent to  $\text{type}(r)$ , then every tree in  $\text{type}(r)$  satisfies  $c''$ , and we can conclude that  $r_i$  is self-disactivating.

**Conjunctions.** For case (a), if the condition of a rule  $r_j$  is of the form

$$l_{j,1} \text{ and } l_{j,2} \dots \text{ and } l_{j,n_j}$$

we can use the tests described in the previous two subsections for conditions that are simple XPath expressions or negations of simple XPath expressions to determine if a rule  $r_i$  may turn any of the  $l_{j,k}$  from False to True. If so, then  $r_i$  may turn  $r_j$ 's condition from False to True, and may thus activate  $r_j$ .

For case (b), suppose the condition of rule  $r_i$  is of the form

$$l_{i,1} \text{ and } l_{i,2} \dots \text{ and } l_{i,n_i}.$$

There are three possible cases:

1. All the  $l_{i,j}$  are simple XPath expressions. In this case,  $r_i$  will be self-disactivating if each of its actions is a deletion which subsumes one or more of the  $l_{i,j}$ .
2. All the  $l_{i,j}$  are negations of simple XPath expressions. In this case,  $r_i$  will be self-disactivating if each of its actions is an insertion which falsifies one or more of the  $l_{i,j}$ .
3. The  $l_{i,j}$  are a mixture of simple XPath expressions and negations thereof. In this case,  $r_i$  may or may not be self-disactivating.

**Disjunctions.** For case (a), if the condition of a rule  $r_j$  is of the form

$$(l_{1,1} \text{ and } l_{1,2} \dots \text{ and } l_{1,n_1}) \text{ or } (l_{2,1} \text{ and } l_{2,2} \dots \text{ and } l_{2,n_2}) \text{ or } \dots \text{ or } (l_{m,1} \text{ and } l_{m,2} \dots \text{ and } l_{m,n_m})$$

we can use the test for conjunctions described above to determine if a rule  $r_i$  may turn any of the disjuncts

$$l_{k,1} \text{ and } l_{k,2} \dots \text{ and } l_{k,n_k}$$

from False to True. If so, then  $r_i$  may turn  $r_j$ 's condition from False to True and may thus activate  $r_j$ .

For case (b), suppose the condition of rule  $r_i$  is of the form

$$(l_{1,1} \text{ and } l_{1,2} \dots \text{ and } l_{1,n_1}) \text{ or } (l_{2,1} \text{ and } l_{2,2} \dots \text{ and } l_{2,n_2}) \text{ or } \dots \text{ or } (l_{m,1} \text{ and } l_{m,2} \dots \text{ and } l_{m,n_m})$$

Then  $r_i$  will be self-disactivating if it leaves False all the disjuncts of this condition. This will be so if

1. all the  $l_{j,k}$  are simple XPath expressions and  $r_i$  disactivates all the disjuncts of its condition as in case 1 for conjunctions above; or
2. all the  $l_{j,k}$  are negations of simple XPath expressions and  $r_i$  disactivates all the disjuncts of its condition as in case 2 for conjunctions above

In all other cases,  $r_i$  may or may not be self-disactivating.

## 5 Conclusions

In this chapter we discussed the provision of ECA rules for XML repositories. We reviewed ECA rules in conventional active databases and highlighted the main new issues that arise in the context of XML data. We described the design of a language for ECA rules on XML, described a prototype implementation, and presented techniques for analysing the behaviour of ECA rule sets defined in our language.

It would be straightforward to extend our language to also support REPLACE events and actions. A REPLACE event would have the syntax

REPLACE  $e$

while a REPLACE action would have the syntax

REPLACE  $e$  BY  $r$

where  $e$  is a simple XPath expression and  $r$  is a simple XQuery expression. The meaning of a REPLACE action is that the set of nodes identified by  $e$  (and their descendants) should be replaced by the XML fragments denoted by  $r$ . For example, the pair of actions in the rule  $r_1$  in Example 1 could be replaced by the single action

```

REPLACE $delta/.../.../high;
BY      <high>$delta/text()</high>

```

Combinations of our analysis techniques for INSERT and DELETE actions could be applied to derive triggering and activation information for REPLACE events.

For future work there are several directions to explore:

- a. There is as yet no accepted standard update language for XML. If ECA rules are to be supported on XML repositories, then whatever standard eventually emerges, there is also the parallel issue of designing the event language to match up with this update language. In this chapter we have done this in the context of our particular update language for XML. We have also shown how triggering and activation relationships can be detected for our particular ECA rules. In general, the ability to analyse ECA rule sets needs to be balanced against their complexity and expressiveness, and this issue also needs to be borne in mind in future developments in ECA rule languages for XML.
- b. We would like to explore more deeply the expressiveness and complexity of the ECA language that we have defined. For example, what types of XML Schema constraints can be enforced and repaired using rules in this language?
- c. In general, `updateDB` in Sect. 3.2 will undertake a set of updates on the repository. For INSERT actions, this may result in nondeterminism in the order in which a set of new fragments are inserted under a common parent, since the BEFORE and AFTER constructs only specify the ordering of new fragments with respect to the existing document content. It is an area of further work to extend our ECA language to capture ordering relationships between new fragments being inserted into a document.
- d. At present our language supports only semantic triggering, though it would be easy to extend it to also support syntactic triggering. Similarly, although we currently assume immediate coupling mode for event/condition and condition/action, it would straightforward to also allow rules with the full range of other coupling modes. However, the practical applicability and performance implications of these extensions is an area that requires further detailed investigation.
- e. We would like to further develop and gauge the effectiveness of our rule analysis and optimisation techniques. For example, incorporating knowledge that certain documents within the repository are valid with respect to a document type definition (DTD) or XML Schema specification may be useful in at least two ways. First, this knowledge can allow us to simplify the XPath expressions used within ECA rules [38]. Second, it can help to obtain more precise type information when doing type inference, which in turn can allow more precise information on triggering and activation dependencies to be inferred.
- f. A related issue is to develop techniques for determining whether a set of ECA rules is type-safe, in other words, whether execution of the rules ensures that each document remains valid with respect to its DTD or XML Schema.
- g. We would like to improve our current prototype implementation, and in particular to integrate our ECA Rules Engine with existing XML repositories, once this technology has become mature enough.



- h. An important issue is to evaluate the applicability and scalability of our language, its execution model and implementation. For this, we are planning to deploy it for providing reactive functionality on distributed RDF repositories of educational metadata, as part of the EU-funded SeLeNe project (see <http://www.dcs.bbk.ac.uk/selene>).

The SeLeNe project is investigating the technical requirements, and possible technical solutions, for ‘self e-learning networks’. A self e-learning network (SeLeNe) will have a peer-to-peer topology. Each peer will manage part of the overall distributed metadata, possibly with replication across peers. This metadata will be expressed in RDF which, if stored as XML, will be amenable to direct manipulation by our XML ECA rule language. Support of such networks will require:

- techniques for reconciliation and integration of metadata describing heterogeneous distributed learning objects (LOs)
- definition of personalised views over this distributed metadata resource
- detection and notification of changes to the LO metadata descriptions
- publish/subscribe functionality

These requirements have a good fit with the functionality that could potentially provided by ECA rules and will provide a challenging testbed for application, evaluation and extension of our language and its implementation, including an opportunity to explore the applicability and performance of a variety of rule coupling modes, and to gauge the effectiveness of our analysis and optimisation methods.

The SeLeNe project will also provide an opportunity to assess the impact of moving from a centralised to a distributed environment, with the additional challenges of network delay, network reliability, tolerance of delays and failures, synchronisation of event detection and action execution, maintaining consistency of the distributed information resource, etc. Some of the challenges of event-based systems in distributed heterogeneous environments are discussed in the chapters by Jacob et al. and Buchmann et al. in this book.

Many of the above open questions would make suitable PhD research topics, for example the questions raised in (a), (c), (e) and (h). Possible Masters-level projects would include (b), (d), (f) and (g).

## References

1. S. Abiteboul, S. Cluet, G. Ferran, and M.-C. Rousset. The Xyleme project. *Computer Networks*, 39:225–238, 2002.
2. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The Lorel query language for semistructured data. *VLDB Journal*, 1(1):68–88, 1997.
3. S. Abiteboul, V. Vianu, B. S. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *JCSS*, 61(2):236–269, 2000.
4. A. Adi, D. Botzer, O. Etzion, and T. Yatzkar-Haham. Push technology personalization through event correlation. In *Proc 26th Int. Conf. on Very Large Databases*, pages 643–645, 2000.

5. A. Aiken, J. Widom, and J. M. Hellerstein. Static analysis techniques for predicting the behavior of active database rules. *ACM TODS*, 20(1):3–41, 1995.
6. J. Bailey and A. Poullovassilis. An abstract interpretation framework for termination analysis of active rules. In *Proc. 7th Int. Workshop on Database Programming Languages, LNCS 1949*, pages 249–266, Kinloch Rannoch, Scotland, 1999.
7. J. Bailey and A. Poullovassilis. Analysis of functional active databases. In P.M.D. Gray et al., editor, *The Functional Approach to Data Management: Modeling, Analyzing and Integrating Heterogeneous Data*. Springer, Berlin Heidelberg New York, 2003.
8. J. Bailey, A. Poullovassilis, and P. Newson. A dynamic approach to termination analysis for active database rules. In *Proc. 1st Int. Conf. on Computational Logic (DOOD stream), LNCS 1861*, pages 1106–1120, London, 2000. Springer, Berlin Heidelberg New York.
9. J. Bailey, A. Poullovassilis, and P.T. Wood. An Event-Condition-Action Language for XML. In *Proc. WWW'2002*, pages 486–495, Hawaii, 2002.
10. J. Bailey, A. Poullovassilis, and P.T. Wood. Analysis and optimisation for event-condition-action rules on XML. *Computer Networks*, 39:239–259, 2002.
11. E. Baralis, S. Ceri, and S. Paraboschi. Improved rule analysis by means of triggering and activation graphs. In T. Sellis, editor, *Rules in Database Systems, LNCS 985*, pages 165–181. Springer, Berlin Heidelberg New York, 1995.
12. E. Baralis, S. Ceri, and S. Paraboschi. Compile-time and runtime analysis of active behaviors. *IEEE Transactions on Knowledge and Data Engineering*, 10(3):353–370, 1998.
13. E. Baralis and J. Widom. An algebraic approach to rule analysis in expert database systems. In *Proceedings of the 20th International Conference on Very Large Databases*, pages 475–486, Santiago, Chile, 1994.
14. E. Baralis and J. Widom. An algebraic approach to static analysis of active database rules. *ACM TODS*, 25(3):269–332, 2000.
15. Michael Benedikt, Wenfei Fan, and Gabriel M. Kuper. Structural properties of XPath fragments. In *Proc. 9th Int. Conf. on Database Theory, LNCS 2572*, pages 79–95, Berlin, 2003.
16. A. Bonifati, D. Braga, A. Campi, and S. Ceri. Active XQuery. In *Proc. of the IEEE Conference on Data Engineering (ICDE)*, 2002.
17. A. Bonifati, S. Ceri, and S. Paraboschi. Active rules for XML: A new paradigm for e-services. *VLDB Journal*, 10(1):39–47, 2001.
18. A. Bonifati, S. Ceri, and S. Paraboschi. Pushing reactive services to XML repositories using active rules. In *Proc. 10th World-Wide-Web Conference*, 2001.
19. S. Ceri and P. Fraternali. *Designing Database Applications with Objects and Rules: The IDEA Methodology*. Addison-Wesley, 1997.
20. S. Ceri, P. Fraternali, and S. Paraboschi. Data-driven one-to-one web site generation for data-intensive applications. In *Proc. 25th Int. Conf. on Very Large Databases*, pages 615–626, 1999.
21. S. Chakravarthy. Architectures and monitoring techniques for active databases: An evaluation. *Data and Knowledge Engineering*, 16(1):1–26, 1995.
22. S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data and Knowledge Engineering*, 14(1):1–26, 1994.
23. E. Cho, I. Park, S. J. Hyum, and M. Kim. ARML: an active rule mark-up language for heterogeneous active information systems. In *Proc. RuleML 2002*, Sardinia, June 2002.
24. S. Cluet, P. Veltri, and D. Vodislav. Views in a large scale XML repository. In *Proc. 27th Int. Conf. on Very Large Databases*, pages 271–280, 2001.
25. A. Couchot. Improving the refined triggering graph method for active rules termination analysis. In *Proc. BNCOD 2002, LNCS 2405*, pages 114–133, Sheffield, 2002.

26. N. Gehani, H. V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model and implementation. In *VLDB'92*, pages 327–338, 1992.
27. Haruo Hosoya and Benjamin C. Pierce. XDuce: A typed XML processing language (preliminary report). In *Proc. WebDB 2000: Int. Workshop on the Web and Databases*, pages 111–116, 2000.
28. H. Ishikawa and M. Ohta. An active web-based distributed database system for e-commerce. In *Proc. Web Dynamics Workshop, London, 2001*. <http://www.dcs.bbk.ac.uk/webDyn/>.
29. A. Karadimce and S. Urban. Refined triggering graphs: A logic based approach to termination analysis in an active object-oriented database. In *Proc. ICDE'96*, pages 384–391, New Orleans, 1996.
30. K. Keenoy et al. Self e-Learning Networks. See <http://www.dcs.bbk.ac.uk/selene/reports/Del22.pdf>, August 2003. SeLeNe Project Deliverable 2.2.
31. K. Kulkarni, N. Mattos, and R. Cochrane. Active database features in SQL3. In N. Paton, editor, *Active Rules in Database Systems*, pages 197–219. Springer, Berlin Heidelberg New York, 1999.
32. Frank Neven and Thomas Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *Proc. 9th Int. Conf. on Database Theory, LNCS 2572*, pages 315–329, Berlin, 2003.
33. N. Paton, editor. *Active Rules in Database Systems*. Springer, Berlin Heidelberg New York, 1999.
34. J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient matching for web-based publish/subscribe systems. In *Proc 7th Int. Conf. on Cooperative Information Systems (CoopIS'2000)*, pages 162–173, 2000.
35. I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating XML. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 413–424, 2001.
36. G. Wagner. How to design a general rule markup language? In *Invited talk at the Workshop XML Technologien für das Semantic Web (XSW 2002)*, Berlin, June 2002.
37. J. Widom and S. Ceri. *Active Database Systems*. Morgan-Kaufmann, 1995.
38. Peter T. Wood. Containment for XPath fragments under DTD constraints. In *Proc. 9th Int. Conf. on Database Theory, LNCS 2572*, pages 300–314, Berlin, 2003.
39. World Wide Web Consortium. XML Path Language (XPath), Version 1.0. See <http://www.w3.org/TR/xpath>, November 1999. W3C Recommendation.
40. World Wide Web Consortium. XSL Transformations (XSLT), Version 1.0. See <http://www.w3.org/TR/xslt>, November 1999. W3C Recommendation.
41. World Wide Web Consortium. XQuery 1.0: An XML Query Language. See <http://www.w3.org/TR/xquery>, November 2002. W3C Working Draft.
42. World Wide Web Consortium. XQuery 1.0 and XPath 2.0 Formal Semantics. See <http://www.w3.org/TR/query-semantics>, November 2002. W3C Working Draft.
43. World Wide Web Consortium. Document Object Model (DOM) Level 3 Core Specification. See <http://www.w3.org/TR/DOM-Level-3-Core/>, February 2003. W3C Working Draft.

---

# Active XQuery

Angela Bonifati<sup>1</sup> and Stefano Paraboschi<sup>2</sup>

<sup>1</sup> ICAR-CNR, Via P.Bucci, 41C - 87036 Rende CS, Italy  
bonifati@icar.cnr.it

<sup>2</sup> Università di Bergamo, Via Marconi, 5 - 24044 Dalmine BG, Italy  
parabosc@unibg.it

**Summary.** We analyze some of the issues arising when an event-condition-action (ECA) rule mechanism is introduced within an XML management system. Apart from the presentation of a specific solution, the focus of the chapter is on two aspects. First, we analyze the coupling modes (immediate and deferred) that can be adopted to integrate the event application part and the rule evaluation part. Then, we illustrate how the component responsible of updating XML data and the rule processing engine can be integrated in a loose or tight way, producing two distinct rule execution semantics.

## 1 Introduction

The Web is currently migrating from a rather static asset to a dynamic configuration, in which automatized mechanisms have an increasing role. In particular, in several applications, such as negotiation portals, billing desks, reservation sites, monitoring services, index directories and many others, the promising XML technology needs to be augmented with special active features. By ‘active’ we mean whatever piece of code is activated synchronously or asynchronously by an occurred event. The events can be general enough to include data modification events or simple invocations of a given function. We have identified many commonalities between this new active breed and traditional active database rules [8, 9, 18]. First, to locate the necessary information, they both need suitable querying capabilities. Next, like stored procedures and active rules, the data need to reside close to the active code, which exploits it to perform its computations. In previous work [3, 4, 5], we showed that traditional triggers, once reshaped in any current XML language, may respond to these needs. Thus, in this chapter, we focus on the definition of an active language for XQuery [11], the standard XML query language.

The addition of active features to Web languages creates many challenges and is reflected in many initiatives. Indeed, many efforts in the industry and in the research community aim at achieving a great level of automatization in a variety of mechanisms. Among them, we cite some commercial ones, such as Macromedia MX [17], Apache Jelly [2], Sun JSP [16], PHP [19] and so on. They strictly integrate the functionality

of a programming language or a script language with XML pages. These approaches are very similar to ours in the fact that an event-condition-action paradigm can be simulated. A specialized interpreter is needed each time an active feature is desired, and this requires some effort and customization. With Active XQuery, we propose a natural extension of the query processor to support triggers for XML data. As we will see while illustrating the architecture, the modifications of the query processor are not traumatic and once done, allow us to exploit the full expressibility of the query language.

In a special application of XML triggers, such as that of monitoring Web pages, one could argue that continuous queries can be fruitfully employed. The main advantage of the trigger solution resides in the possibility of programming a reaction to changes only when needed, leading to savings in the query instantiations and in the client-server connections. Moreover, triggers are located where the documents reside, and these do not need to be previously fetched in memory as for enacting a continuous query. Migration of the current XML query languages toward support of the ECA paradigm is quickly feasible with little effort from a syntactic point of view. Compared with traditional technologies used to implement Web services (e.g. agents), active rules can be preferred for their simplicity. Active rules will not be used as the final tool for each kind of e-service, because several brand-new e-services are too complex to be implemented by means of triggers and require more complex implementation mechanisms (such as using the agent technology in workflow management systems). However, triggers are inherently suitable for the rapid development of several conventional applications, such as view maintenance, constraint handling, business rules, etc. In addition, active rules can use as their action methods and procedures (compliant to any kind of an XML-compatible Web protocol, like SOAP [21]) that are executed remotely and implemented elsewhere. As a final observation, triggers are installed inside the interested Web servers. Conversely, in an agent-based architecture, the agents are software modules instructed to go around in the network and to visit different hosts. Therefore, attacks to protected sites or execution of malicious code are more dangerous in an agent-based scenario.

A similar approach to ours has been developed within the Active XML project [1]. Active XML proposes a new framework for Web services development in which function calls are embedded into XML by using specialized *sc* tags. Calls are shipped on a remote server by means of a SOAP wrapper, and a description of the Web service in the *Web Services Description Language* (WSDL [27]) is available to distribute the service signature around the network. XQuery is used to express service calls as well as any programming language.

We organize this chapter as follows: in Sect. 2 we present the event model of Active XQuery and the choices of rule execution; in Sect. 3 we present some introductory examples of Active XQuery; in Sect. 3.1 we describe the syntax; and in Sect. 4 we describe the two alternative semantics, comparing their features. In Sect. 5 we show a classification of XML triggers and show, through examples, some other important application fields.

## 2 Active XQuery Event Base

XQuery is the standard query language for XML documents developed by the W3C. It comes with a wide list of features for retrieving and re-constructing data within the XML documents, and with additional special functions for manipulating the text inside the data [24] or for treating special data types [28]. However, none of the manipulation operations, such as those issued by an update language, have yet been incorporated in the language. For this reason, in defining an active language for XQuery, we rely on the update model for XQuery recently developed in a research paper [23]. The update operations only affect the event model of the trigger language and the set of actions that are allowed in the action clause. Any other update language can be adopted instead of that in [23] for Active XQuery, by only modifying the syntax. Indeed, the semantics we have defined is general enough to explain the interpretation of any update operation.

Before illustrating the syntax and the execution model of Active XQuery, we briefly present the event model, the types of events and the event consumption modes. These are indeed the additional ingredients that need to be added to the query language in order to prepare the active extension.

### 2.1 Event Model

An event is something that happens at a point in time such that it is relevant to properly react to it. The event model permits us to determine which is the event context, and how event occurrences are captured. Events are relatively easy to capture when they occur as a result of users' operations, either as update operations or plain queries: we name these events *explicit*. Indeed, in many settings the user may interact with an interface to submit his/her desired update operations. These operations cannot be always captured at run time, and thus the modifications that occurred can only be grasped from a direct comparison of document versions. For instance, the operations over the document may be inferred by applying a diff algorithm [13] or a diff tool [15, 14], which compares two different snapshots of the same document. For conciseness, these events are called *derived* and they are stored into event sequences, which we call *event traces*.

Given an update language, its update commands may describe both types of events, *explicit* and *derived*. In some sense, the way events are collected or computed is orthogonal to the definition of the active language, which is the aim of this chapter. However, it may impact the behavior of the rules, as shown in our previous work [6].

As a final observation, the concept of event in the XML world is different from the one defined for tuples or for objects in the relational and object-oriented models. *Order* may be critical for documents. In the XML native data model [7], elements are ordered and attributes are unordered. However, if XML is used as an interchange format, order is less relevant than in other application fields. Relevance of order decreases as far as the focus is on data rather than on presentation.

This makes things more complicated and leads to two alternative interpretations of events, according to an *order-dependent* or *order-independent* semantics, along with the distinction between positional and nonpositional operations. The *order-dependent* semantics defines a correct sequence of events that eventually produces a version of the document that depends on the order. Positional operations help to guarantee an order-dependent semantics. With an *order-independent* semantics, all the versions are equivalent to any that contains the same items in a different order. Nonpositional operations are allowed in such a case. We do not distinguish between an order-dependent and an order-independent event semantics, because we consider the ordered semantics of the update language and directly transpose this semantics in the rule language. Thus, the sequence of operations in the update language enforces the order of events in the rule language evaluation.

## 2.2 Immediate Versus Deferred Event Consumption

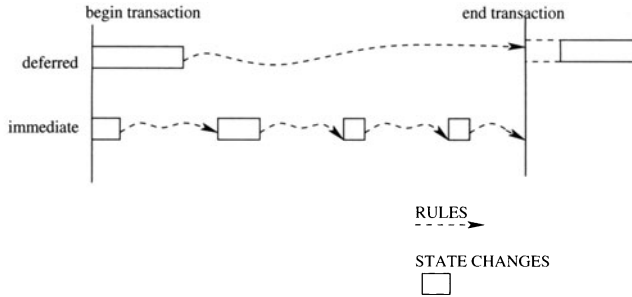
When a triggered rule is considered or actually executed, the events responsible for the triggering may undergo different treatments, which are called *event consumption modes*. Usually, the event consumption time is associated with the actual execution of rules rather than on condition evaluation. In our case, event consumption modes can be *immediate* or *deferred*.<sup>3</sup> When events are explicit, one can decide to process them (and to activate the corresponding rules) or to keep them for later evaluation. When events are derived from the cross-comparison of two versions of the document, events processing (and rule execution) is naturally postponed to a subsequent time. Precisely, the explicit event detection supports the immediate and deferred event consumption modes, while the derived event detection only supports the *deferred* event consumption.

Therefore, in the *immediate* mode, events are consumed exactly when they occur (or, equivalently, rules are fired). The *immediate* mode of consumption enforces a sequence of state changes and rule sessions tightly interleaved (Fig. 1). The state changes correspond to the sessions in which events are detected and rule sessions are the intervals in which events are consumed. In the *deferred* mode, events are consumed (rules are fired) promptly at the end of the transaction or at a given rule assertion point beyond the end of the transaction (as graphically represented in Fig. 1). In such a case, the sequence of state changes and rule sessions are loosely interleaved. When an error occurs in rule evaluation, the last rule session is rolled back in the immediate mode and the entire transaction is rolled back in the deferred mode.

The event consumption mode also affects the rule execution mode and determines two ways to evaluate the triggers.

- The *immediate* execution mode processes the triggers exactly when events (due to update operations) occur.

<sup>3</sup> In active databases, the terms *immediate* and *deferred* are used to characterize the alternatives in the coupling mode between the event and the condition/action of rules. In this chapter we associate with the terms a different interpretation.



**Fig. 1.** Sequence of state changes and rule execution sessions in the immediate and deferred event consumption modes

- In the *deferred* execution mode, rules are not processed at the earliest opportunity after events, but in a separate session, and by analyzing the occurred event trace off-line.

In our discussion, we show the *immediate* execution mode. Nevertheless, a *deferred* execution mode has a similar treatment.

Compared to relational updates, XQuery updates can be seen as *bulk* statements, since they may involve arbitrarily large fragments of documents, which are inserted or dropped by means of a single statement. These may trigger active rules that monitor events relative to internal portions of such document fragments. Thus, the main difficulty in extending the notion of triggers from the relational domain to the XML domain is indeed due to the different granularity between update events and rule events. To overcome this difficulty we have defined a first algorithm that expands bulk statements into a collection of equivalent statements, each one relative to a smaller fragment of the documents, so as to guarantee that any trigger defined for the document will be correctly considered. Each of these statements is in turn a self-standing XQuery update. This first algorithm defines a *loosely bundling immediate semantics*.

A second algorithm is defined that permits us to expand the original update statement into a unique expanded statement. Rules and updates are more tightly interleaved than in the previous case, since only one expanded statement is generated. This second algorithm defines a *tightly bundling immediate semantics*.

### 3 Active XQuery by example

Let us assume a scenario based on the following `Library.xml` document, which belongs to the XML repository of a university library and describes the books stored on the shelves:

```
<Library>
...
```



```

<Shelf nr="45">
  <Book id="AO97">
    <Author> J. Acute </Author>
    <Author> J. Obtuse </Author>
    <Title> Triangle Inequalities </Title>
  </Book>
  <Book id="So98">
    <Author> A. Sound </Author>
    <Title> Automated Reasoning </Title>
  </Book>
  ...
</Shelf>
...
</Library>

```

An example of an update to the library is the bulk insertion of a whole shelf (nr. 45) into the document by means of the following XQuery update statement (s0). The new library content is extracted from a collection of new shelves, located in a separate document (within the repository). In order to insert fragment \$frag, the language requires us to envelop the actual INSERT operation into an external UPDATE clause, targeted to a variable that is bound to the element that will *contain* the fragment (node \$target). Recursively nested update statements (and therefore UPDATE clauses) are allowed within the curly brackets.

```

s0:
FOR $target IN document("Library.xml")/Library,
  $frag IN document("New.xml")/Shelves/Shelf
WHERE $frag/@nr="45"
UPDATE $target { INSERT $frag }

```

In our scenario, the library automatically maintains an index with a list of all authors, keeping pointers (IDREFs) to the library entries. The following XML excerpt demonstrates the author index:

```

<AuthorIndex>
  ...
  <AuthorEntry uni="PoliMi" pubs=".. AO97 ..">
    <Name> J. Acute </Name>
    <PubsCount> ... </PubsCount>
  </AuthorEntry>
  ...
  <AuthorEntry uni="Princeton" pubs=".. So98 ..">
    <Name> A. Sound </Name>
    <PubsCount> ... </PubsCount>
  </AuthorEntry>
  ...
</AuthorIndex>

```

Triggers are responsible to guarantee referential integrity among the authors' publications (pubs attribute) and the books in the library. In particular, we want to guarantee the following properties:

- **No dangling references:** *Deletion of a Book element will cause all its authors (listed in the index part of the document) to lose their ‘dangling’ references to that publication.*
- **Automatic indexing:** *Insertion of a Book element will cause a new reference to be inserted into all index items that represent new book’s authors. Note that this may require a new Author element to be inserted into the list.*

Automatic deletion of dangling pointers is performed by trigger NoDangle, which updates AuthorEntry elements removing from their pubs attributes all references<sup>4</sup> to the deleted book (identified by keyword OLD\_NODE):

```
CREATE TRIGGER NoDangle
AFTER DELETE OF document("Library.xml")//Book
FOR EACH NODE
DO ( FOR
    $AutIndex IN document("Library.xml")//AuthorIndex,
    $MatchAut IN $AutIndex/AuthorEntry
        [Name = OLD_NODE/Author],
    $DangRef IN $MatchAut/ref(pubs, OLD_NODE/@id)
UPDATE $AutIndex { DELETE $DangRef } )
```

Two other triggers perform the insertion of new references and new Author Entry elements. If one of the authors of the new book is not yet in the list, the higher-priority trigger AddNewEntry inserts a new ‘empty’ AuthorEntry element. Thus, low-priority trigger AddNewReference can assume that the index already contains entries for all the incoming authors.

```
CREATE TRIGGER AddNewEntry
AFTER INSERT OF document("Library.xml")//Book
FOR EACH NODE
LET $AuthorsNotInList := (
    FOR $n IN NEW_NODE/Author
    WHERE empty(//AuthorIndex/AuthorEntry[Name=$n])
    RETURN $n )
WHEN ( not( empty($AuthorsNotInList) ) )
DO ( FOR $ai IN document("Library.xml")//AuthorIndex,
    $NewAuthor IN $AuthorsNotInList
UPDATE $ai
    { INSERT <AuthorEntry>
        <Name> {$NewAuthor/text()} </Name>
        <PubsCount> 0 </PubsCount>
    </AuthorEntry> } )

CREATE TRIGGER AddNewReference
WITH PRIORITY -10
AFTER INSERT OF document("Library.xml")//Book
```

---

<sup>4</sup> Note that bindings to a single IDREF within an IDREFS attribute are declared according to the syntax extension proposed in [23].

```

FOR EACH NODE
DO ( FOR $ai IN document("Library.xml")//AuthorIndex,
    $a IN $ai/AuthorEntry[Name=$a]
    UPDATE $a
    { INSERT new_ref(pubs, NEW_NODE/@id)} )

```

Finally, triggers `IncrementCounter` and `DecrementCounter` maintain a counter of authors' publications (we only show `IncrementCounter` for brevity).

```

CREATE TRIGGER IncrementCounter
AFTER INSERT OF //new_ref(pubs)
FOR EACH NODE
LET $Counter := NEW_NODE/../PubsCount
DO ( FOR $AuthorEntry IN NEW_NODE/../
    UPDATE $AuthorEntry
    { REPLACE $Counter WITH $Counter + 1 } )

```

These triggers demonstrate that the execution of the action part of a trigger can cause the activation of other triggers (`AddNewReference` triggers `IncrementCounter`).

### 3.1 Quick Syntax of Active XQuery

The simplified syntax of an XQuery trigger is the following:

```

CREATE TRIGGER Trigger-Name
[WITH PRIORITY Signed-Integer-Number]
(BEFORE | AFTER)
(INSERT | DELETE | REPLACE | RENAME)+
OF XPathExpression ( , XPathExpression)*
[FOR EACH (NODE | STATEMENT)]
[XQuery-Let-Clause]
[WHEN XQuery-Where-Clause]
DO (XQuery-UpdateOp | ExternalOp
| TransactCom)

```

- The `CREATE TRIGGER` clause is used to define a new XQuery trigger with the specified name.
- Rules can be prioritized in an absolute ordering, expressed with an optional `WITH PRIORITY` clause, which admits as argument any signed integer number. If this clause is omitted, the default priority is zero.
- The `BEFORE/AFTER` clause expresses the triggering time relative to the operation.
- Each trigger is associated with a set of update operations (insert, delete, rename, replace) adopted from the update extension of XQuery [23].

- The operation is relative to elements that match an XPath expression (specified after the `OF` keyword), i.e. a step-by-step path descending the hierarchy of documents (according to [12] and its update-related extensions<sup>5</sup>). One or more predicates (XPath *filters*) are allowed in the steps to eliminate nodes that fail to satisfy given conditions. Once evaluated on document instances, the XPath expressions result in sequences of nodes, possibly belonging to different documents.
- The optional clause `FOR EACH NODE/STATEMENT` expresses the trigger granularity. A *statement-level* trigger executes once for each set of nodes extracted by evaluating the XPath expressions mentioned above, while a *node-level* trigger executes once for each of those nodes. Based on the trigger granularity, it is possible to mention in the trigger the transition variables:
  - If the trigger is node-level, variables `OLD_NODE` and `NEW_NODE` denote the affected XML element in its before and after state.
  - If the trigger is statement-level, variables `OLD_NODES` and `NEW_NODES` denote the sequence of affected XML elements in their before and after state.
- An optional *XQuery-Let-Clause* is used to define XQuery variables whose scope covers both the condition and the action of the trigger. This clause extends the ‘REFERENCING’ clause of SQL:1999 because it can be used to redefine transition variables.
- The `WHEN` clause represents the trigger condition and can be an arbitrarily complex XQuery *where* clause. If omitted, a trigger condition that specifies `WHEN TRUE` is implicit.
- The action is expressed by means of the `DO` clause, and it can be accomplished through the invocation of an arbitrarily complex update operation. In addition, a generic *ExternalOp* syntax indicates the possibility of extending the XQuery trigger language with support to external operations, e.g. permitting it to send mail or to invoke SOAP procedures. A *TransactCom* syntax indicates a generic transaction command, such as a *fullRollback*, a *partialRollback* or a *commit* of the current transaction in which the trigger is executed.

For a complete syntax of XQuery refer to [10]. For the syntax of the update language, refer to [23].

## 4 The Immediate Execution Mode for Active XQuery

### 4.1 Intuitive Semantics

The semantics of XQuery triggers should be as close as possible to the semantics of SQL:1999 triggers [20, 22], as already discussed. Accordingly, each XQuery operation should be computed in the context of a recursive procedure, such that:

- At the time of execution of an update, the set of affected nodes is computed (leading to the evaluation of transition variables).

<sup>5</sup> The additional keyword `ref`, introduced in [23], can be used to denote a single IDREF within an attribute of type IDREFS.

- A given update statement is preceded by BEFORE triggers and followed by AFTER triggers;<sup>6</sup> statement and node-level triggers may interleave and are considered in priority order.
- If a given trigger executes an operation and this in turn causes some triggerings, the trigger execution context is suspended, and a new procedure is recursively invoked. The depth of recursion is limited by some given threshold, which is system specific.

However, such intuitive semantics cannot be immediately replicated in XQuery, given the hierarchical structure of XML and the ‘bulk’ nature of update primitives. According to the update language of [23], the insertion of ‘content’ may refer to an arbitrarily large XML fragment, and likewise the deletion of a node may cause the dropping of an arbitrarily large XML fragment. By ‘bulk’ operations, we mean those operations of the update language comprising two or more constituents. The concept will be illustrated through an example.

*Example 1.* Consider the following update statement, which inserts a whole XML fragment containing subelements and attributes:

```
s0:
FOR $target IN document("Library.xml")/Library,
    $frag IN document("New.xml")/Shelves/Shelf
WHERE $frag/@nr="45"
UPDATE $target { INSERT $frag }
```

The XML excerpt that is inserted is the following:

```
<Shelf nr="45">
  <Book id="A097">
    <Author> John Acute </Author>
    <Author> Jack Obtuse </Author>
    <Title> Applying Triangle Inequalities </Title>
  </Book>
  <Book id="So98">
    <Author> Anthony Sound </Author>
    <Title> Principles of Automated Reasoning </Title>
  </Book>
  ...
</Shelf>
```

Within *s0* there is one bulk statement `INSERT $frag` that needs to be expanded over the children of *Shelf*.

Expansion of bulk updates is the first step to semantics definition. We will show in the following section how expansion is accomplished. Note that the SQL language instead supports update operations targeted to a given table and does not need to expand original statements. Therefore, a precise description of the semantics of XQuery must be combined with carefully defined management of bulk updates.

<sup>6</sup> In order to avoid nondeterministic and/or nonmonotonic behavior, BEFORE triggers may be subject to limitations in their actions.

## 4.2 Update Expansion

A first strategy (named *loosely binding semantics* (LBS)) with bulk updates consists of decomposing each original XQuery bulk update  $\mathfrak{s}_0$  into a decomposition sequence  $S$  of smaller granularity self-standing updates, such that the change to each XML element involved in the update is addressed by a specific update operation of  $S$ . This strategy requires the definition of two separate mechanisms, one for expanding updates and one for executing them, where the latter includes the composition of updates with triggers. Note that update expansion requires access to the XML document; this is obvious in the case of bulk deletion (when the specific elements to be deleted need to be first accessed), but occurs with bulk insertions as well.

An alternative strategy is possible, consisting of interleaving update expansion and trigger evaluation. We call this semantics *tightly binding semantics* (TBS), as opposed to the *loosely binding semantics* aforementioned. In this strategy, the original XQuery update statement  $\mathfrak{s}_0$  is decomposed into a single nested update statement. In such a case, composition of updates with triggers is stricter, since triggers are invoked within the bulk update execution.

We have devised two algorithms addressing the two strategies. We will show both of them in this chapter, but we will focus on the first semantics for the implementation of the Active XQuery system. In designing the expansion strategy (in both semantics), we use a visit of the hierarchical structures that mimics the ‘natural’ order of update propagation, in which inserts proceed top-down, and deletions proceed bottom-up. Such a visit strategy is described, in a simple case of bulk insert, in Fig. 2 for the loosely binding semantics (the adopted notation displays attributes as black circles, elements as empty circles and PCDATA content as empty boxes); the considered fragment is that of the `Shelf` excerpt considered in the previous section.

Following this semantics, fragments are visited in a mixed breadth–depth order. In the first step of the algorithm, the first-level elements are visited and grouped into a common update statement. Root nodes need to be treated separately, since they lack a common ancestor. This is a precise choice with respect to the different semantics of sets introduced earlier in this chapter. Here, the sets depend directly on the expansion strategy. Figure 2 shows that four self-standing statements are obtained from the expansion of  $\mathfrak{s}_0$ . The order of these statements is indicated by means of the progressive numbers.

When switching to the second semantics (Fig. 3), one can notice that the number of involved groups increases. The elements are visited in a depth-first order, and separate groups correspond to separate update substatements. With this second strategy,  $\mathfrak{s}_0$  remains a single statement, albeit nested. The smaller updates are grouped together taking apart the PCDATA content (which is considered in a separate update). The major difference from the first strategy is in the number of the smaller updates that are treated separately in nested substatements. For example, as represented in Fig. 2, the second-level elements are inserted with a separate statement. Conversely, in Fig. 3, the second-level elements are part of the same statement but inserted at a different time (the attribute and one element first and then, after the substatements from 4 to 10, the last element).

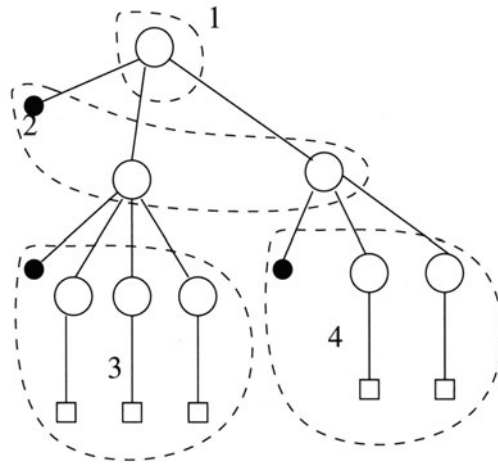


Fig. 2. Visit of an XML fragment (LBS)

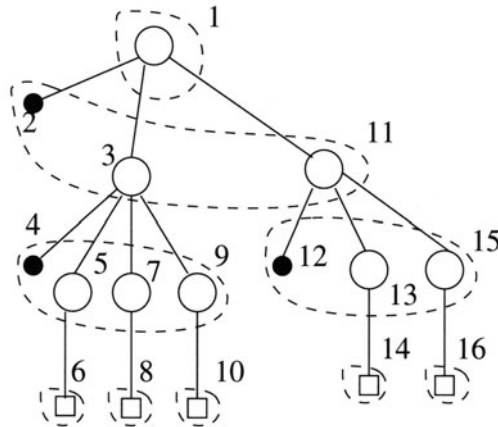


Fig. 3. Visit of an XML fragment (TBS)

### 4.3 Expansion Algorithm (Loosely Binding Semantics)

In the following we detail the expansion process for `INSERT` statements. `DELETE` statement expansion is analogous (and is omitted for brevity), while `REPLACE` statements can be rewritten in terms of `DELETE` statements immediately followed by suitable `INSERT` statements. `RENAME` operations perform mere name changes and do not require further expansion.

Essentially, expansion proceeds with an intermixed depth-first and breadth-first visit of the involved fragments. In particular, all the operations relative to the insertion of XML nodes with a common father are enveloped in the same update statement. Among these nodes (which can be attributes, elements or `PCDATA` content), attributes are inserted first, then elements and `PCDATA` content are inserted in their proper order.

Elements are inserted as empty couples of tags if they contain a complex structure (and thus require further expansion). Otherwise they solely have PCDATA content, and they are inserted together with such content.

Function *buildContentOfFirstUpdate* addresses the construction of the first expanded statement, targeted to the same variable as the user update operation (referenced in the external UPDATE clause). It takes the bulk statement *ST* as argument and outputs one update statement that inserts all the roots of the involved fragments. Thereafter, the algorithm starts from each complex root node and expands its complex subelements. This expansion is entrusted to function *expandNode*, which is then recursively invoked on all complex subelements of these elements.

**Algorithm 1 Bulk Statement Expansion.** *Given an arbitrary bulk XQuery update statement ST, the algorithm returns UL, an ordered list of XQuery expanded update statements and directives.*

*In a preprocessing phase, the algorithm computes the required data structures. Precisely, variable V represents the argument of the user update clause; variable S contains the bindings to the involved fragments; FClause and WClause are the for and where clauses of ST; XFClause is an ad hoc clause that is repeatedly used in the construction of the results; cur\_path is a variable that is assigned to the current path, as soon as it is available by the visit of the fragments; variable name is used to name the generated statements and match them with their corresponding directives.*

*This version of the algorithm takes care of the expansion of insert statements and accepts for simplicity only flat user statements. Possible nested user updates can be treated via previous reduction to a list of flat statements.*

```

begin
  V = getUpdateVariable(ST)
  S = bindInvolvedFragments(ST)
  FClause = getFORClause(ST)
  WClause = getWHEREClause(ST)
  XFClause = FClause + ", $curFrag IN " + V +
    "/*[empty(" + V + "/*[AFTER $curFrag]]]"
  cur_path = "$curFrag"
  name = buildUniqueName(cur_path)
  UL = "EvalBefore(" + name + ") " + "Name:" + name + " "
  UL += FClause + WClause
  UL += "UPDATE " + V + "{ "
  UL += buildContentOfFirstUpdate(ST) + "}"
  for each fragment in S, consider again its root node N:
    if ( N is complex and requires further expansion )
      then UL += expandNode(N, cur_path, XFClause, WClause)
    UL += "EvalAfter(" + name + ") "
  return UL
end;

```

**FUNCTION** *expandNode(Node N, String cur\_path,*  
*String XFClause, String WClause)*  
**RETURNS OUT,** *an ordered list of update statements and directives*



```

begin
  name = buildUniqueName(cur_path)
  OUT = "EvalBefore(" + name + ") " + "Name:" + name + " "
  if( cur_path="$curFrag" )
    then OUT += XFClause + WClause +
      "UPDATE $curFrag { "
    else OUT += XFClause + ", $cur_node IN " + cur_path +
      WClause + "UPDATE $cur_node { "
  for each attribute A of N
    OUT += "INSERT new_attribute( "
    OUT += A.name + ", " + A.value + ") "
  for each subelement C of N
    if( C is an XML-Element )
      then if( C has only PCDATA content )
        then OUT += "INSERT " + buildTagWithPCDATA(C)
        else OUT += "INSERT " + buildEmptyTag(C)
      else OUT += "INSERT " + C.content
    OUT += "}"
  for each subelement C of N:
    if( C is complex and requires further expansion )
      then cur_path += "/" * [ " + position of C + " ]
      OUT += expandNode(C, cur_path, XFClause, WClause)
    OUT += "EvalAfter(" + name + ") "
  return OUT
end;

```

Moreover, the algorithm interleaves the statements with special directives to the rule engine that enable the construction of conflict sets. In particular, the directive *EvalBefore* contains the name of the statement that it precedes, and the directive *EvalAfter* contains the name of the statement that it follows. The positions of these directives within the list of statements reflect the intrinsic semantics of the original statement. If we consider an INSERT operation, the *EvalBefore* directives precede each expanded statement, while *EvalAfter* directives solely follow the expanded statement of the leaf portions of the fragment. The *EvalBefore* directives that refer to the remaining (nonleaf) portions are postponed by recursion and follow the directive of the last leaf of the fragment, as shown below.

*Example 2.* We consider the expansion of the bulk statement *s0* (from Sect. 4.1), which inserts an entire shelf in the document *Library.xml*. We need to expand *s0* into smaller self-standing update statements, in order to make the defined triggers sensitive to the insertion of the children of the *Shelf* element. The expansion algorithm outputs the following sequence:

```

EvalBefore(s1)

Name:s1
FOR $x IN document("Library.xml")/Library,
  $frag IN document("New.xml")/Shelves/Shelf[@nr="45"]

```

```

UPDATE $x
{ INSERT <Shelf/> }

EvalBefore(s2)

Name:s2
FOR $x IN document("Library.xml")/Library,
  $frag IN document("New.xml")/Shelves/Shelf[@nr="45"],
  $curfragment IN $x/*[empty($x/*[AFTER $curfragment])]
UPDATE $curfragment
{ INSERT new_attribute(nr, "45")
  INSERT <Book/>
  INSERT <Book/> }

EvalBefore(s3)

Name:s3
FOR $x IN document("Library.xml")/Library,
  $frag IN document("New.xml")/Shelves/Shelf[@nr="45"],
  $curfragment IN $x/*[empty($x/*[AFTER $curfragment])],
  $cur_node IN $curfragment/*[1]
UPDATE $cur_node
{ INSERT new_attribute(id, "AO97")
  INSERT <Author> J. Acute </Author>
  INSERT <Author> J. Obtuse </Author>
  INSERT <Title> Triangle Inequalities </Title> }

EvalAfter(s3)

EvalBefore(s4)

Name:s4
FOR $x IN document("Library.xml")/Library,
  $frag IN document("New.xml")/Shelves/Shelf[@nr="45"],
  $curfragment IN $x/*[empty($x/*[AFTER $curfragment])],
  $cur_node IN $curfragment/*[2]
UPDATE $cur_node
{ INSERT new_attribute(id, "So98")
  INSERT <Author> A. Sound </Author>
  INSERT <Title> Automated Reasoning </Title> }

EvalAfter(s4)

EvalAfter(s2)

EvalAfter(s1)

```

The order of evaluation of triggers corresponds to the intuitive order that could be expected by a user unaware of the decomposition process. This can be appreciated if

one considers the hierarchy  $s_1 < s_2 < \{s_3, s_4\}$  and the order of execution of before and after triggers. For instance, the ‘AFTER INSERT’ triggers that are triggered by  $s_2$  (referred to by the directive `EvalAfter( $s_2$ )`) see the side effects not only of statements  $s_3$  and  $s_4$ , but also of all trigger executions caused by them.

#### 4.4 Expansion Algorithm (Tightly Binding Semantics)

We synthetically present here the alternative expansion algorithm (which we have not adopted) describing a deeper fusion between the update statement and the triggers. This solution becomes eligible when the rule engine can be built on top of the query optimizer. The number of directives is greater than in the first expansion and is equal to the number of elements and of attributes of the fragment (see Fig. 3). For these reasons, in the remainder of this chapter we will focus on the first kind of expansion strategy.

**Algorithm 2 Bulk Statement Expansion (2).** *Given an arbitrary bulk XQuery update statement centered on node  $N$ , the algorithm returns  $EXP$ , a single XQuery expanded update statement, enriched with directives for rule evaluation.*

*This version of the algorithm does not need a preprocessing phase, since the initial FOR clause, WHERE clause and update variables are replicated only once in the final statement. The algorithm merely computes the variable  $X$  attached to the root node  $N$  and calls `expandNode`, a recursive function on it. The function descends in the children of the root node.*

*The underlying hypothesis assumes that there is one root for each update statement, which is exactly what happens in the adopted update language.*

*This version of the algorithm is also focused on the expansion of insert statements and for simplicity accepts only flat statements. Possible nested updates can be treated via previous reduction to a list of flat statements.*

```

begin
  X = getUpdateVariable(N);
  EXP = expandNode(N, 1, 0);
end;
FUNCTION expandNode(Node N, int counter, int position)
RETURNS ONE, a single expanded XQuery update statement
begin
  ONE = "EvalBefore(" + N.name + ")"
  ONE += "INSERT" + buildEmptyTag(N)
  if isRoot(N)
    ONE += "FOR $V1 IN " + X +
           "/*[empty(" + X + "/*[AFTER $V1]]"+ "UPDATE $V1{"
  else
    ONE += "FOR $V" + counter + " IN $V" + counter-1
           "+ /*[" + position + "] UPDATE $V"+counter+"{"
  for (int i=0; i < N.getAttributesNumber(); i++)

    ONE += "EvalBefore(" + N.getAttribute(i).name + ")"
    ONE += "INSERT new_attribute(" + N.getAttribute(i).name+
           ", " + N.getAttribute(i).value + ")";
    ONE += "EvalAfter(" + N.getAttribute(i).name + ")"

```

```

for (int i=0; i < N.getChildrenNumber(); i++)

    if (N.isElement())
        ONE+= expandNode(N.getChild(i), counter+1, i+1)
    else
        ONE+= " INSERT " + N.getChild(i).content + " ";

ONE+= " } ";
ONE+= " EvalAfter(" + N.name + " ) "
return ONE
end;

```

The output of the expansion is longer than the one provided by the first algorithm. However, the update statement is unbroken. For lack of space, we omit here the update expanded according to the second algorithm; it can be retrieved from the book Web site.

#### 4.5 Trade-off Analysis Between the LBS and TBS Semantics

The loosely binding and the tightly binding semantics have been described through their respective algorithms for expansion of bulk updates into elementary updates. In this section, we compare them taking into account some aesthetic and functional criteria.

- In the loosely binding model, expanded updates are autonomous and separate. Triggering operations are computed for each update and involved triggers occur exactly when the update ends its execution. When operating with the tightly binding model, instead, the original update operation does not lose its integrity, and involved triggers need to be coalesced with the ongoing update execution.
- Separation of triggers and updates in the loosely binding semantics leads to a separation between the trigger engine and the query evaluator. Indeed, this permits us to obtain a quick implementation of a trigger mechanism upon the existing querying technology. In the tightly binding semantics, this separation is not feasible since there is one monolithic update on which triggers are invoked. As shown in the previous point, in such a case triggers evaluation is not detached from the update evaluation; hence, the trigger engine and the query engine need to be unified.
- As with SQL:1999, the set of nodes affected by an update is computed before computing the update by the decomposer, i.e. according to the semantics of the original, user-level update primitive. This happens for both semantics.
- In the loosely binding expansion, the obtained update statements are self-standing and contain heavily duplicated path expressions. This is due to the need to isolate the context of the operation. In the tightly binding semantics, replication is avoided since the update is unbroken and a single context specification is exploited. In practice, in the loosely binding semantics, this disadvantage is minor,

because transparent caching mechanisms should make such traversals very efficient; moreover, simple optimizations could be done, such as preserving pointers to already computed nodes from one statement execution to the next one.

- While the LBS expansion yields a sequence of well-defined and compact statements, in the TBS semantics expansion leads to a single statement that is rather lengthy and verbose. Moreover, while the LBS is more readable, the second expansion seems to be more natural and intuitive.
- The LBS decomposition of a bulk statement into a sequence of statements leads to exposing intermediate states – the ones left by a prefix of the decomposition sequence. Therefore, the decomposition strategy affects the *semantics* of triggers, as this in turn is order dependent. This is inevitable: order dependence characterizes even relational systems and is amplified by a hierarchical structure that can be processed in many ways. In the TBS decomposition, this problem is eliminated since the update evaluation is intertwined with the triggers invocation. In such a case, however, the technology of the rule engine has to be heavily modified in order to be combined with query execution.

	LBS semantics	TBS semantics
Evaluation of triggers and updates	Separate	Integrated
Separation between trigger and query engines	Feasible	Not feasible
Computation of set of nodes	Before decomposition	Before decomposition
Repetition of the same path traversals	High	Low
Decomposition features	Intermediate states	No intermediate states

**Table 1.** Differences and similarities between LBS and TBS semantics

In Table 1, we illustrate the main differences between the two kinds of decompositions. At first glance, note that the TBS semantics is reasonably more intuitive than the LBS semantics. Indeed, the TBS semantics has few path expression replicas and is even more compact. However, to adopt the TBS semantics, the update processing has to be intermixed with rule evaluation, and technology of the rule engine is heavily affected. Because of the results of this trade-off analysis, we chose to deploy the LBS approach since it can be easily supported ‘on top’ of an existing XQuery optimizer, in the same way as a trigger engine can be easily supported on a relational storage system [25]. Therefore, in the remainder of this section, we focus on the execution mode for LBS-expanded updates and XQuery triggers.

#### 4.6 Description of Trigger Execution Mode

Given the update decomposition algorithm, we can now define the trigger execution mode precisely, thus giving an operational semantics of the combined execution

of updates and triggers. As observed, our query execution mode is inspired by the SQL:1999 execution mode, however it is adapted to the hierarchical nature of XML and exploits the expansion of statements. Assume the XQuery engine is starting the execution of a generic bulk update statement *S*, which has been submitted by the user. The following algorithm defines this semantics operationally.

```

PROCEDURE EXECUTE_STATEMENT(Statement S)
1 Call EXPAND_STATEMENT(S) and store the
  returned structures (RF and SIL)
2 For each item Ii in SIL, if it is
2.1 an 'EvalBefore' instruction, call
  COMPUTE_BEFORE_CONFLICT_SET(Sn, RF),
  where Sn is the statement related to Ii
2.2 an 'EvalAfter' instruction, call
  COMPUTE_AFTER_CONFLICT_SET(Sn, RF),
  where Sn is the statement related to Ii
2.3 an update statement, execute Ii,
  updating the XML repository

PROCEDURE EXPAND_STATEMENT(Statement S)
RETURNS FragmentSequence RF,
      StatementInstructionList SIL
1 Retrieve RF, the set of fragments that are
  relevant for the execution of S
2 Expand S into SIL by visiting RF with the
  expansion algorithm
3 Return SIL and RF (NEW_RF and/or OLD_RF)

PROCEDURE COMPUTE_BEFORE_CONFLICT_SET
  (Statement Sn, FragmentSequence RF)
1 Compute BT, the set of eligible
  BEFORE triggers activated by Sn
2 Order all computed triggers according to
  their global ordering
3 For each trigger T in BT, PROCESS_TRIGGER(T)

PROCEDURE COMPUTE_AFTER_CONFLICT_SET
  (Statement Sn, FragmentSequence RF)
1 Compute AT, the set of eligible AFTER
  triggers activated by Sn
2 Order all computed triggers according
  to their global ordering
3 For each trigger T in AT, PROCESS_TRIGGER(T)

PROCEDURE PROCESS_TRIGGER(trigger T)
1 Calculate pointers corresponding to
  NEW_NODE(S) and OLD_NODE(S)
2 If any, evaluate the Let clause of T and bind
  the new variables

```

- 3 Evaluate the condition C of T
- 4 If C evaluates to TRUE, EXECUTE\_STATEMENT(A)
- (A being the action of T)

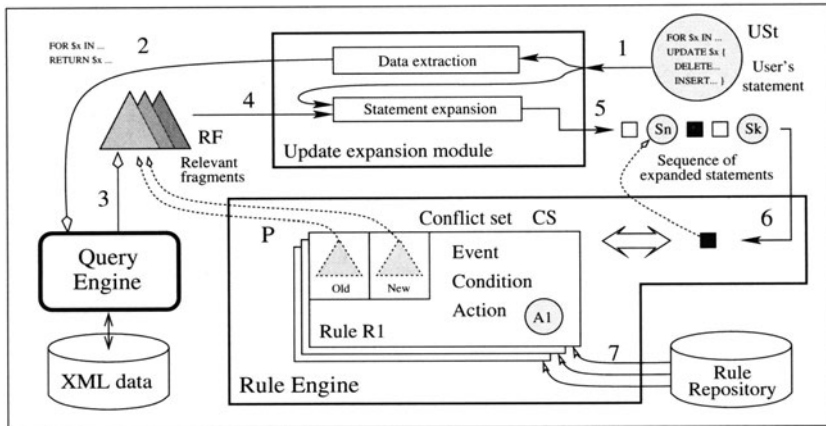


Fig. 4. System architecture

In the former algorithm EXECUTE\_STATEMENT is invoked on  $S$ . The expansion algorithm 1 retrieves the set of relevant fragments (RF, involved by  $S$ ) and produces a sequence of statements and directives (SIL). SIL is generated according to the mixed depth–breadth order visit of the fragments, and this is accomplished by calling procedure EXPAND\_STATEMENT. The algorithm needs to communicate with the query engine in order to inspect XML data and to build RF, which constitutes a separate structure. More precisely, REPLACE operations will yield both NEW\_RF and OLD\_RF structures, deletions will produce only OLD\_RF structures and insertions only NEW\_RF structures. These structures are accessed by rules in order to bind transition variables, as explained below. The last task of EXPAND\_STATEMENT is to return SIL and to rename RF as NEW\_RF and/or OLD\_RF (depending on the type of original statement  $S$ ).

Once the EXPAND\_STATEMENT has been completed, each item  $I_i$  in SIL is one of the following mutually exclusive cases. If the item is an *EvalBefore* directive, then the procedure COMPUTE\_BEFORE\_CONFLICT\_SET is invoked; if it is an *EvalAfter* directive, then the procedure COMPUTE\_AFTER\_CONFLICT\_SET is invoked; otherwise, it is an expanded statement, ready to be executed. Both COMPUTE\_BEFORE\_CONFLICT\_SET and COMPUTE\_AFTER\_CONFLICT\_SET receive as parameters the statement  $S_n$  referred by the directive and RF.

Procedures COMPUTE\_BEFORE\_CONFLICT\_SET and COMPUTE\_AFTER\_CONFLICT\_SET respectively compute BT and AT, the sets of triggered rules. This defines the conflict sets pertaining to statement  $S_n$ , which include both statement-level and node-level triggers, in priority order. For each trigger  $T$  in these conflict sets, PROCESS\_TRIGGER is executed with  $T$  as parameter. Local pointers are calcu-





activation, since many similar nodes can be affected by  $S_n$ , and each rule instance is provided with the two pointers `OLD_NODE(S)` and `NEW_NODE(S)`, that represent the old/new fragment(s) (one of them might be set to null).

All triggered rules are collected in a conflict set (CS), where they are ordered with respect to their priority. Note that node-level and statement-level rules are mixed in CS, and that CS contains rules addressing nodes with different tagnames as well, since  $S_n$  affects all subelements and attributes of a single node. Only the actions of those rules with true condition are performed.

Statement A1 can be a bulk statement itself, and thus it must be processed by a recursive replication of the abstract machine described in Fig. 4. If we imagine a scenario in which the execution of a rule action causes another [cascading] rule to activate, we can represent the resulting stack of execution contexts like in Fig. 5. Here statement USt triggers rule R1, the action A1 of R1 is itself expanded, it triggers rule R2 and an update is eventually performed, since statement  $a_{21}$  causes no rule activation (the conflict sets related to  $a_{21}$  are empty).

## 5 Experiences of Applications of XML triggers

In this section, we focus on the application fields that Active XQuery is able to cover. In Sect. 5.1, we first provide an outline of these fields and their perspectives with respect to the current XML advanced technology.

### 5.1 Summarization of Application Fields

Table 2 summarizes the application fields along two dimensions. The vertical dimension distinguishes between those special-purpose triggers that are implemented ad hoc by the programmer and those that, conversely, are supposed to be produced automatically. The distinction is inspired by the classification provided in [8] for database triggers. The horizontal dimension aims to characterize triggers as integrated in the XML repository, or provided as internal services exportable to the outside, or, alternatively, provided by third parties as external services.

	Internal only XML repository services	Internal XML repository services	External services
Hand-crafted	Metadata management, Document versioning, internal audit trails	Alerters, extenders, audit trails	Web business rules supply chain manag. transactional aervices
Gene- rated	XML Schema identity and domain constraints, XML view maintenance	Generic integrity constraints	B2B process integration (e.g. in WFMS)

**Table 2.** XML trigger applications

Starting from the first column, ‘internal only XML repository services’ are services not exportable to the outside. They basically are devoted to the management of XML data internal to the repository. Handcrafted triggers of this species include those that contribute to build semantic knowledge over the data,<sup>7</sup> document histories and traces of user behavior internal to the repository. Generated triggers are triggers that are supposed to be automatically generated by examining the document schemas or the document views. These are mainly triggers for maintaining XML Schema identity and domain constraints, and triggers for keeping views of XML data consistent with the original documents.

The second column includes those triggers that are programmed within the repository and are instrumental in offering to the outside exportable services or in providing fruitful information. These include handcrafted triggers such as prevalently notification mechanisms, extenders (separate structures, like flat files or specialized indexes that are kept aligned with the XML content) and audit trails. Generated triggers are instead generic integrity constraints that cannot be expressed in XML Schema, and need to be maintained separately. These kinds of constraints are mainly semantic constraints that resemble the assertions of traditional databases. All these can be automatically generated once they have been encoded in a powerful constraint language.

The third column indicates triggers that come from external applications. Handcrafted triggers include business rules that include personalizers, classifiers and alerters centered on documents published on the Internet. Notification mechanisms again fall into this category as they are provided by external applications to deliver business information with rich content. Moreover, transactional services such as reservations or orders and supply chain management can be programmed by means of asynchronous business rules. Finally, generated triggers mostly arise in B2B process integration, such as triggers to be used in workflow enactment. In such a case, triggers can be created automatically as long as there is a high-level conceptual specification of the business process. External applications that follow these criteria (e.g. have an abstract model of rules) are eligible for the automatic generation of concrete triggers.

## 6 Summary and Future Directions

This chapter described Active XQuery and enriched the presentation with a few examples; additional examples can be retrieved from the Web site of the book. We also showed some relevant application fields, which happen to be different from those of traditional database languages. In particular, we argue that active rules can be employed as rapid prototypes for Web services and respond to the novel need of rendering the Web as dynamic as possible. We presented two alternative semantics for XQuery triggers that enable the use of rules in a rich spectrum of applications.

<sup>7</sup> This is a very relevant application field. The Semantic Web Initiative is a W3C outstanding activity attracting the Web community. We feel that active rules may effectively serve as facilities to generate machine-understandable data and put them on the Web.

We highlighted the cases of *immediate* and *deferred* event detection. We specialized the *immediate* semantics for Active XQuery, while noting the fact that the *deferred* semantics has an equivalent evaluation, once the event trace is known.

We showed the *immediate* semantics for Active XQuery. We aimed at compatibility with SQL:1999, in spite of some difficulties because of ‘bulk’ updates and the hierarchical nature of XML. This goal was achieved by defining a rule execution schema based on the preliminary expansion of user-provided update statements. We provided two kinds of expansion, based on LBS and TBS semantics. We showed that the main advantage of the former is the clean separation between the rule engine and the query optimizer, yielding a modular architecture in which the query optimizer can be merely plugged in.

Our study raises several optimization and research issues, briefly discussed below.

- **Schema-driven optimization.** The expansion of bulk statements can benefit from the availability of the documents schema, either expressed as a DTD or in terms of XML Schema. In particular, it is possible to avoid expanding the updates relative to those parts of a document that do not activate rules. Such an optimization is relevant within most XML repositories, as few of their element types correspond to distinctive real-world entities and are therefore targeted to triggers.
- **Internal optimizations.** Specialized indexing techniques can be devised in order to optimize the rule engine for repeated executions of the same query over different fragments, e.g. by adding data structures that point to ‘sibling’ nodes. Such data structures can be set up by the expansion module, as it is aware of both the queries and the fragments. Other indexes may link the nodes of the XML repository to rules in the rule repository, in order to facilitate the extraction of the rules that are triggered by given operations.
- **Limitations of expressive power.** As with the SQL:1999 standard, the class of ‘legal’ BEFORE triggers remains to be defined [20]. A simple solution can be the exclusion from their actions of any update operation or lookup into transition variables, but this in practice limits the expressive power of BEFORE triggers to error signaling. In many cases, however, such lookups and updates do not cause any inconsistency; therefore, the class of ‘legal’ triggers – maybe relative to given XML schemas or documents – remains to be defined.
- **Definition of illegal executions.** Similarly, certain classes of executions are illegal, for instance, when the update performed by a trigger affects the data that caused the triggering, thus yielding to contradictory situations. Such illegal executions should be identified, and execution-time ‘traps’ should be instrumented in order to detect them.
- **Compile-time trigger analysis.** Trigger analysis, applied to a given XML repository and rule set, could be used to detect anomalous behavior, such as the lack of termination or confluence [26]. Conversely, trigger analysis could be used to ‘validate’ triggers, thus excluding that a given rule set could lead to any illegal execution. These techniques are defined for relational triggers but need to be extended for XQuery triggers.

## References

1. S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. In *ACM SIGMOD*, San Diego, CA, June 2003.
2. Jelly: Executable XML. <http://jakarta.apache.org/commons/sandbox/jelly/>.
3. A. Bonifati, D. Braga, A. Campi, and S. Ceri. "Active XQuery". In *Proc. of ICDE 2002*, February 2002.
4. A. Bonifati, S. Ceri, and S. Paraboschi. Active rules for XML: A new paradigm for e-services. In *1st Workshop on E-Services (TES)*, Cairo, Egypt, September 2000.
5. A. Bonifati, S. Ceri, and S. Paraboschi. Pushing reactive services to XML repositories using active rules. In *Proc. of 10<sup>th</sup> World Wide Web Conf.*, pages 633–641, Hong Kong, China, May 2001.
6. A. Bonifati, S. Ceri, and S. Paraboschi. Event trace independence of active behavior, July 2003. Technical Report.
7. T. Bray, J. Paoli, and C. M. Sperberg-McQueen (eds). Extensible Markup Language (XML) 1.0 (2nd Edition). W3C Recommendation, Oct. 2000. <http://www.w3.org/TR/2000/REC-xml-20001006>.
8. S. Ceri, R. J. Cochrane, and J. Widom. Practical applications of triggers and constraints: Success stories and lingering issues. In *Proc. of 26th VLDB Conf.*, pages 254–262, Cairo, Egypt, September 2000.
9. S. Ceri and J. Widom. Deriving production rules for constraint maintenance. In *Proc. of 16th VLDB Conf.*, pages 566–577, Brisbane, Australia, September 1990.
10. D. Chamberlin, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu (eds). "XQuery 1.0: An XML Query Language". W3C Working Draft, June 2001. <http://www.w3.org/TR/2001/WD-xquery-20010607/>.
11. D. Chamberlin et al.(eds.). XQuery: A Query Language for XML. W3C Working Draft, 15 February 2001. <http://www.w3.org/TR/2001/WD-xquery-20010215/>.
12. J. Clark and S. J. DeRose (eds). "XML Path Language (XPath) Version v1.0". W3C Recommendation, April 1999. <http://www.w3.org/TR/xpath>.
13. G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proc. of ICDE*, San Jose, USA, February 2002.
14. Delta XML Tool (Commercial), 2001. <http://www.deltaxml.com/>.
15. IBM Alphaworks XML Diff Tool, 1999. <http://www.alphaworks.ibm.com/tech/xmldiffmerge>.
16. Sun's JavaServer Pages. <http://java.sun.com/products/jsp/>.
17. Macromedia Dreamweaver. <http://www.macromedia.com/>.
18. N. Paton, editor. "Active Rules in Database Systems". Springer, Berlin Heidelberg New York, 1999.
19. PHP. <http://www.php.net/>.
20. K. G. Kulkarni R. Cochrane and N. Mendonça Mattos. Active database features in SQL3. In *Active Rules in Database Systems*, pages 197–219. Springer, Berlin Heidelberg New York, 1999.
21. "Simple Object Access Protocol (SOAP) 1.1 (W3C Note)", 2000. <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.
22. SQL:1999/Foundation. ISO-IEC 9075-2:1999, June 1999.
23. I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. "Updating XML". In *ACM SIGMOD*, Santa Barbara, May 2001.
24. XQuery and XPath Full-text Use Cases. <http://www.w3.org/TR/xmlquery-full-text-use-cases/>.

25. J. Widom. The Starburst active database rule system. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 8(4):583–595, 1996.
26. J. Widom and S. Ceri. *Active database systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.
27. Web Service Description Language (WSDL) 1.1 (W3C Note), 2001. <http://www.w3.org/TR/wsdl>.
28. XML Query Use Cases. <http://www.w3.org/TR/xmlquery-use-cases/>.

---

# Active XML:

## A Data-Centric Perspective on Web Services

Serge Abiteboul<sup>1,4</sup>, Omar Benjelloun<sup>1</sup>, Ioana Manolescu<sup>1</sup>,  
Tova Milo<sup>1,3</sup>, and Roger Weber<sup>2</sup>

<sup>1</sup> INRIA, France

Firstname.Lastname@inria.fr

<sup>2</sup> ETH Zurich, Switzerland

Roger.Weber@inf.ethz.ch

<sup>3</sup> Tel Aviv University, Israel

<sup>4</sup> Xyleme S.A., France

**Summary.** We propose in this chapter a peer-to-peer architecture that allows for the integration of distributed data and Web services. It relies on a language, Active XML, where documents embed calls to Web services that are used to enrich them, and new Web services may be defined by XQuery queries on such active documents. Embedding calls to functions or even to Web services inside data is not a new idea. Our contribution, however, is to turn them into a powerful tool for data and services integration. In particular, the language includes linguistic features to control the timing of service call activations. Various scenarios are captured, such as mediation, data warehousing and distributed computation. A first prototype is also described.

## 1 Introduction

Data integration has been extensively studied in the past in the context of company infrastructures e.g., [23, 36, 46]. However, since the Web is becoming one of its main targets, data integration has to deal with large-scale issues, and faces new problems of heterogeneity and interoperability between ‘loosely-coupled’ sources, which often hide data behind programs. These issues have been recently addressed in two complementary ways. First, major standardization efforts have addressed and partially resolved some of the heterogeneity and interoperability problems via (proposed) standards for the Web, like XML, SOAP and WSDL [43]. XML, as a self-describing semistructured data model, brings flexibility for handling data heterogeneity,<sup>5</sup> while emerging standards for *Web services* like SOAP and WSDL simplify the interoperability problem by normalizing the way programs can be invoked over the Web. Second,

---

<sup>5</sup> Complementary standards for meta data, like RDF, also address semantic heterogeneity issues.

the increasingly popular peer-to-peer architectures seem to provide a promising solution for the problems coming from the independence and autonomy of sources and the large scale of the Web. Peer-to-peer architectures provide a decentralized infrastructure in sync with the spirit of the Web and that scales well to its size, as demonstrated by recent applications [34, 27].

We believe that the peer-to-peer approach, together with the aforementioned standards, form the proper ground for data integration on the Web. What is still lacking, however, is the ‘glue’ between these two paradigms. This is precisely what is provided by Active XML (AXML, in short), the topic of this chapter, a declarative framework that harnesses XML and Web services for data integration, and is put to work in a peer-to-peer architecture.

The AXML framework is centered around *AXML documents*, which are XML documents that may contain calls to Web services. When calls included in a document are fired, the latter is enriched by the corresponding results. In some sense, an AXML document can be seen as a (partially) materialized view integrating plain XML data and dynamic data obtained from service calls. It is important to stress that AXML documents are syntactically valid XML documents. As such, they can be stored, processed and displayed using existing tools for XML. Since Web services play a major role in our model, we also provide a powerful means to create new ones: they can be specified as queries with parameters on AXML documents, using XQuery [53], the future standard query language for XML, extended with updates.

Documents with embedded calls were already proposed before (see related work in Sect. 2 for a detailed account). But AXML is first to actually turn them into a powerful tool for data integration, by providing the following features:

**Controlling the activation of calls and the lifespan of data.** By giving means to declaratively specify when the service calls should be activated (e.g., when needed, every hour, etc.), and for how long the results should be considered valid, our approach allows us to capture and combine different styles of data integration, such as warehousing, mediation and flexible combinations of both.

**Services with intentional input/output.** Standard Web services exchange ‘plain’ XML data. We allow AXML Web services to exchange AXML data that may contain calls to other services. Namely, the arguments of calls as well as the returned answers may include both extensional XML data and intentional information represented by embedded calls. This allows us to delegate portions of a computation to other peers, i.e., to distribute computations. Moreover, the decision to do so can be based on considerations such as peer capabilities or security requirements.

**Continuous services.** Existing Web services either use a (one-way) messaging style or a remote procedure calls (RPC) style – they are called with some parameters and (eventually) return an answer. But often, a *continuous* behavior is desired, where a (possibly infinite) stream of answers is returned for a single call. As an example, consider subscription systems where new data of interest is pushed to users (see, e.g., [35]). Similarly, a data warehouse receives streams of updates from data sources, for maintenance purposes. Streams of results are also generated, for instance, by sensors (e.g., thermometers or video surveillance cameras). The AXML framework

adds this capability to Web services and allows for the use and creation of such continuous Web services.

These features are combined in a peer-based architecture, illustrated by Fig. 1. Each *peer* contains a repository of AXML documents, some AXML Web services definitions and an Evaluator module that acts both as a client and as a server:

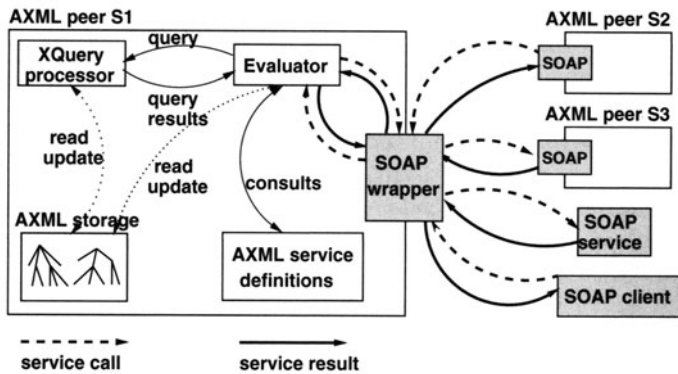


Fig. 1. Outline of the AXML data and service integration architecture

**Client** The Evaluator acts as a client by choosing which of the service calls embedded in the AXML documents need to be fired at each point in time, firing the calls and integrating the returned answers into the documents. It is important to stress that any Web service can be used, be it provided by an AXML peer or not, as long as it complies with standard protocols for Web service description and invocation [39, 47].

**Server** The Evaluator acts as a server by accepting requests for the AXML services supported by the peer, executing the services (i.e., evaluating the corresponding XQuery) and returning the result.

Observe that since AXML services query AXML documents and can accept (resp. return) AXML documents as parameters (resp. results), a service execution may require the activation of other services calls. Thus, these two tasks are closely interconnected.

The chapter is organized as follows: After an overview of related work, the AXML language is presented in Sect. 3, mainly through an extended example. A formal semantics for AXML documents and services is presented in Sect. 4, while security and peer capabilities are considered in Sect. 5. An evaluation strategy and an implementation are discussed in Sect. 6. The last section is the conclusion and points to some interesting open problems.

## 2 Related Work

Active XML touches several areas in data management and Web computing. We next briefly consider these topics and explain how AXML relates to them.



**Basic technologies/standards.** The starting point of the present work is the semistructured data model and its current standard incarnation, namely XML [50]. We rely primarily on an XML query language and on protocols for enabling remote procedure calls on the Web. Disparate efforts to define a query language for XML are now unifying in the XQuery proposal [53], and its subset XPath [52]. We use these as the basis of our service definition language. As for remote procedure calls, the various industrial proposals for Web services, e.g., .NET by Microsoft, e-speak by HP, Sun One by Sun Microsystems, are also converging toward using a small set of specifications to achieve interoperability.<sup>6</sup> Among those, we are directly concerned with the SOAP and WSDL, which are used in our implementation.

SOAP [39] is an XML-based protocol that lets applications exchange information over the Web. It takes advantage of its widely spread protocols (such as HTTP or SMTP) and provides simple conventions for issuing/answering function calls. It also defines standard ways to serialize common programming language constructs, such as arrays and compound types.

WSDL [47] is an XML format for describing Web services, in terms of the operations they provide and their binding to actual protocols. Most important for us, WSDL precisely defines the types of the input and output parameters of each operation, using XML Schema.

More indirectly, UDDI registries [41] can be used by our system, e.g., to publish or discover Web services of interest.

**Data integration.** Data integration systems typically consist of data sources, which provide data, and mediators or warehouses, which integrate it with respect to a schema. The relationship between the former and the latter is defined using a global-as-view or local-as-view approach [21, 29]. While mediators/warehouses can also serve as data sources for higher integration levels, a clear hierarchy between the data providers typically exists. In contrast, all Active XML peers play a symmetric role, both as data sources and as partially materialized views over the other peers, thus enabling a more flexible and scalable network architecture for data integration. Web services are used as uniform wrappers for heterogeneous data sources, but also provide generic means to apply various transformations on the retrieved data. In particular, tools developed for schema/data translation and semantic integration, e.g., [29] can be wrapped as Web services and used to enrich the Active XML framework.

**Services composition and integration.** Integration and composition of Web services (and programs in general) have been an active fields of research [44]. Intentional data was used in MultiLisp [24], under the form of ‘futures’, i.e., handles to results of ongoing computations, to allow for parallelism. Ambients [13, 12], as *bounded* spaces where computation happens, also provide a powerful abstraction for process and devices mobility on the Web. More recently, standard languages have even been proposed in the industry, like IBM’s Web Services Flow Language [48] or Microsoft’s XLang [49] for specifying how Web services interact with each other and how messages and data can be passed consistently between business processes and service interfaces.

<sup>6</sup> An organization was even created for this purpose; see <http://www.ws-i.org>.

While AXML allows us to compose services, to use the output of some service calls as input to others, and even to define new services based on such (possibly recursive) compositions, the focus here is not on workflow and process-oriented techniques [15] but on *data*. AXML is not a framework for service composition, but for data integration using Web services.

Our formal foundation is based on nondeterministic fixpoint semantics [5], which was primarily developed for Datalog extensions. In that direction, the chapter has also been influenced by recent works on distributed Datalog evaluation [26].

**Embedded calls.** As already mentioned, the idea of embedding function calls in data is not a new one. Embedded functions are already present in relational systems [42], e.g., as stored procedures. Method calls form a key component of object databases [14]. The introduction of service calls in XML documents is thus very natural. Indeed, external functions were present in Lore [32], and an extension of object databases to handle semistructured data is proposed in [28], thereby allowing one to introduce external calls in XML data. Our work is tailored to XML and Web services. In that sense, it is more directly related to Microsoft Smart Tags [38], where service calls can be embedded in Microsoft Office documents, mainly to enrich the user experience by providing contextual tools. Our goal is to provide means of controlling and enriching the use of Web service calls for data integration, and to equip them with a formal semantics.

**Active databases and triggers.** The activation of service calls is closely related to the use of triggers [42] in relational databases, or rules in active databases [45]. Active rules were recently adapted to the XQuery context [10]. A recent work considered firing Web service calls [11]. We harness these concepts to obtain a powerful data integration framework based on Web services. In some sense, the present work is a continuation of previous works on *ActiveViews* [1]. There, declarative specifications allowed for the automatic generation of applications where users could cooperate via data sharing and change control. The main differences from *ActiveViews* are that (i) AXML promotes peer-to-peer relationships versus interactions via a central repository, and (ii) the cornerstones of the AXML language are XPath, XQuery and Web services versus object databases [14].

**Peer-to-peer computing.** Peer-to-peer computing is gaining momentum as a large-scale resource sharing paradigm by promoting direct exchange between equal-footed peers [27, 34]. We propose a system where interactions between peers are at the core of the data model, through the use of service calls. Moreover, it allows peers to play different roles and does not impose strong constraints on interaction patterns between peers, since they are allowed to define and use arbitrary Web services. While we do not consider in this chapter issues such as dynamic data placement and replication, or automatic peer discovery, we believe that solutions developed in the peer computing community for these problems (see, for instance, [34]) can benefit our system, and we plan to investigate them in future work.

### 3 AXML by Example

In this section, we introduce *Active XML* via an example. In Sect. 3.1, we present a simple syntax for including service calls within AXML documents and outline its core features. Section 3.2 deals with intentional parameters and results of service calls. We then consider, in turn, the lifespan of data, the activation of calls and the definition of AXML services.

#### 3.1 Data and Simple Service Integration

At the syntactic level, an AXML document is an XML document. At the semantic level, we view an AXML document as an *unordered data tree*, i.e., the relative order of the children of a node is immaterial. While order is important in document-oriented applications, in a database context like ours it is less significant and we assume that, if needed, it may be encoded in the data. Also, we attach a special interpretation to particular elements in the AXML document that carry the special tag `<sc>`, for *service call*; these elements represent service calls that are embedded in the document.<sup>7</sup> In general, a peer may provide several Web services. Each service may support an array of functions. We use here the terminology *service call* for a call to one of the functions of a service.

As an illustration, consider the AXML document corresponding to my personal page for auctions that I manage on my peer, say `mypeer.com`. This simple page contains information about categories of auctions I am currently interested in, and the current outstanding auction offers for these categories. The page may be written as follows:

```
<myAuctions> Auctions I'm interested in.
  <category name="Toys">
    <sc>auction.com/getOffers("Toys")</sc>
  </category>
  <category name="Glassware">
    <sc>eBay.com/getAuctions("Glassware")</sc>
  </category>
</myAuctions>
```

While the category names are explicitly written in the document, the offers are specified only *intentionally*, i.e., using service calls instead of actual data. Here, the list of toy auctions is provided by `auction.com`. On that server, the function `getOffers`, when given as input the category name `Toys`, returns the relevant list of offers, as an XML document. The latter is merged in the document, which may now look as follows:

```
<myAuctions> Auctions I'm interested in.
```

---

<sup>7</sup> For readability, we use a simple syntax for `<sc>` elements. The complete syntax is omitted here.

```

<category name="Toys">
  <sc>auction.com/getOffers("Toys")</sc>
  <auction aID="1">
    <description>Stuffed bear toy</description>
  </auction>
  <auction aID="2">
    ...
  </category>
  ...
</myAuctions>

```

Observe that the new data is inserted as sibling elements of `<sc>`, and that the latter is *not* erased from the document, since we may want to reactivate this call later to obtain new auction offers. Finally, note that in the case of a *continuous* service, several result messages may be sent by the service for one call activation. In this case, all the results accumulate as siblings of the `<sc>` element.

**Merging service results.** More refined data integration may be achieved using ID-based data fusion, in the style of ,e.g., [6, 19, 37]. In XML, a DTD or XML Schema may specify that certain attributes uniquely identify their elements. When a service result contains elements with such identifiers, they are merged with the document elements that have the same identifiers, if such exist. To illustrate this, assume that `auction.com` supports a `getBargains` function that returns the list of the ten currently most attractive offers, each one with its special bargain price. Suppose also that `aID` is a key for `auction` elements. If an auction element with `aID` '1' is returned by a call to `getBargains`, the element will be 'merged' with the auction with `aID` '1' that we already have.

**XPath service parameters.** The parameters of a service call may be defined extensionally (as in the previous examples) or may use XPath queries. For instance, the `getOffers` service used above gets as input a category name. Rather than giving the name explicitly, we can specify it intentionally using a relative XPath expression [52]:

```

<myAuctions> Auctions I'm interested in.
  <category name="Toys">
    <sc>auction.com/getOffers([../@name/text()])</sc>
    ...
  </category> ...
</myAuctions>

```

The XPath expression `../@name/text()` navigates from the `<sc>` node to the parent `<category>` element, and then to its `name` attribute. The service is then called with the `name` attribute's value as a parameter. In this example, there is only one possible instantiation for the XPath expression. In general, several document subtrees may match a given XPath expression. There are two possible choices here. Either to activate the service several times, once per each possible instantiation, or alternatively to call it just once, sending the forest consisting of all the instantiations a single parameter. In our implementation, we took the first approach as a default, but in

principle one could add an attribute to the `<sc>` element to explicitly specify which one of the two semantics is preferred for a particular call. Besides the parameters, the name of the called peer as well as the name of the service itself may be specified using relative XPath expressions. The same default semantics applies.

### 3.2 Intentional Parameters and Results

The parameters of the services calls that we have seen so far were (instantiated as) simple strings. In general, the parameters of a service call may be arbitrary AXML data, specified either explicitly, or by an XPath expression. In particular, AXML parameters may contain calls to other services, leading thus to *intentional* service parameters. For example, to get a more adequate set of auctions, we may use a service that, in addition to the category name, needs the current user budget, which is itself obtained by a call to the bank services:

```
<myAuctions> Auctions I'm interested in.
  <category name="Toys">
    <sc>auction.com/getOffers1([../@name/text()]),
      <sc>bank.com/getBudget("Bob")</sc>
    </sc>
  </category>
</myAuctions>
```

Up to now, we have not discussed where and when a service call is activated. In the above example, we already face a choice concerning the activation of `getBudget`. We may call it first, and then call `getOffers` providing it with the result. Another option is to call directly `getOffers` with the intentional parameter and let it handle the activation of the call to `getBudget`. We discuss this issue in Sect. 5.

Services may not only get intentional data (i.e., AXML documents with embedded service calls) as input, but also return such data as a result. As an example, each auction in the result of `getOffers` may contain a call to a `getDetails` service that provides more information about that particular auction:

```
<myAuctions> Auctions I'm interested in.
  <category name="Toys">
    <sc>auction.com/getOffers([../@name/text()])</sc>
    <auction aID="1">
      <description>Stuffed bear toy</description>
      <sc>auction.com/getDetails([../@aID])</sc>
    </auction>...</category>...
</myAuctions>
```

Observe that intentional results already appear in practice in many popular applications. For example, the Google search engine returns, for a given keyword, some document URLs plus (possibly) a handle for obtaining more answers. With this handle, one can obtain a new list and perhaps another handle. Therefore, AXML service

calls can be seen as a generalization of HTML hyperlinks, that handles calls to Web services.

### 3.3 Controlling the Lifespan of Data

So far, all the service call results were accumulated in the calling document. In practice, we need more flexibility to manage these results, so that we may replace old results with new ones, discard data that became too old or inconsistent, etc. Many models and techniques have been proposed for managing data lifespan, particularly in the fields of version management, temporal databases and active databases. For our purposes, we chose a suitable simple model that may be extended with more complex features.

To manage data lifespan, we conceptually attach a special attribute `expiresOn` to any data node in an AXML document.<sup>8</sup> Some nodes may have an explicit expiration time whereas others will inherit it from one of their ancestors. Expired nodes should simply be viewed as erased from the document.

The value of the `expiresOn` attribute is an event that may depend on time and/or on the document content. For example, if a user wants to specify that her interest in a product category lasts only until February 19th, 2003, then the element will have the following form:

```
<category name="Toys" expiresOn="Feb. 19th, 2003">...
```

Data returned by a service may also come with an expiration time specification. This is a very useful feature that allows a service provider to state how long the particular result is meaningful. For example, `getOffers` may inform the user of an auction's closing time by setting `expiresOn` for the returned data. The lifespan of a service call result may be explicitly *overwritten* by the caller. This is done using a `valid` attribute in the `sc` element. The `valid` attribute can be a function of the time when the call was (last) answered, denoted *rt*. For example, the following call states that auctions in Category *A* are archived for one year.

```
<sc valid="rt + 1 year">auction.com/
  contGetOffers("A")
</sc>
```

### 3.4 Controlling the Activation of Calls

To control *when* a service call is activated, we use two attributes of `<sc>` elements, namely *mode* and *frequency*. The value of the `frequency` attribute is similar to that of `valid`, except that it is a function of *ct*, the time when the service was last called. Thus, we can easily specify a given instant, a time interval, the occurrence of an event, etc. By default, a service is called only once, when the document is registered. We

<sup>8</sup> Strictly speaking, it is not possible in XML to attach an attribute to a `#PCData` node. It is possible to do so in AXML.

say that a call has *expired* when, according to its frequency attribute, it should be activated.

The mode of a call is either *lazy* or *immediate*. In immediate mode, the call is activated when it expires; if the call is in lazy mode, the fact that it has expired only means that the service has to be called whenever the data produced by this call is needed (e.g., by a query over the AXML document).

The data validity and the call activation mode and frequency together provide a flexible and powerful tool for capturing various integration scenarios. This is illustrated next. In the following integration styles, the first three assume regular (noncontinuous) services, whereas the last one relies on continuous services:

- mediator style: set `valid` to 0 (note that an immediate mode would not be meaningful in this case)
- mediator style with caching: choose a nonzero value for `valid`, and lazy mode
- warehousing mode with pulling information: `valid` larger than `frequency` and immediate mode
- warehousing mode with pushing: choose a nonzero value for `valid`, and immediate mode

Note that the activation of a call is dissociated from the lifespan of its results. For example, if we wish to call `getOffers` every day, and keep the results for a week after their acquisition, we would write:

```
<sc valid="rt + 1 week" frequency="ct + 1 day">
  auction.com/getOffers("Toys")
</sc>
```

*Remark 1. (Timeout).* In the case of a noncontinuous service, it may happen that the answer returns very late, or never returns at all. In practice, it is useful to have *timeouts* for calls. When the timeout is reached, the system abandons hope of getting the result. An exception-handling mechanism should also be provided to manage such events.

### 3.5 AXML Service Definition

The AXML framework allows us to call arbitrary Web services and also to define new ones, as illustrated in this section. In short, an AXML service is defined by a parametrized XQuery query over the peer's AXML documents. As an example, `getOffers`, which returns all currently open auctions for a given category, may be defined at `auction.com` as follows:

```
let sc auction.com/getOffers($c) be
  for $cat in document("auction.com/a.xml")//category,
    $a in $cat/auction,
    $aID in $a/@aID/text(),
    $des in $a/description/text()
```

```

where $cat/@name/text()=$c
return <auction aID={$aID}>
      <description>{$des}></description>
      <sc>auction.com/getDetails({../@aID})</sc>
</auction>

```

In the above example, the category parameter  $\$c$  is of type #PCDATA (text). The query consults an AXML document ( $a.xml$ ), which may contain service calls, and constructs an AXML document with some calls (e.g., to the `getDetails` function of `auction.com`). Here again we face a choice concerning the activation of `getDetails`: we may call it first, and only then return the answer of `getOffers`. Another option is to return the document immediately and let the caller of `getOffers` handle the activation of the calls to `getDetails`. The particular type of the service result may be described by an XML Schema [51], as advised by the WSDL specification [47]. This information can be used to choose between the two options mentioned above.

To define *continuous* AXML services, we simply prefix the definition with the keyword `continuous`. Thus, a continuous variant of a `getOffers`, returning the set of interesting auctions whenever it changes, is defined as follows: `let continuous sc auction.com/contGetOffers($c) be...` Additional parameters can be defined to specify the frequency of updates, and whether to send full results or deltas. They are not detailed here.

To define AXML services with side effects, in the absence of a standardized language for XML updates, we use the extension to XQuery proposed in [40].

### 3.6 Discussion

We conclude this section with two remarks regarding consistency and termination.

**Consistency.** We assume that the document we start with is well-formed and obeys its DTD (or XML Schema), if one is specified for it. An inconsistency may arise if one call leads to constructing a document that no longer obeys the schema. While some of this may be prevented by consulting the declared signature of the used services [8, 25], static type checking becomes more complicated because of the use of ID-based element fusion and of XPath expressions in call parameters.

**Termination and recursion.** We have seen above that a service call may return intentional answers. Note that this may lead to a nonterminating computation: the result of a service call may contain new service calls that need to be activated. Those in turn may return new calls to be activated, and so on. Similarly, the processing of a particular service call (namely, the evaluation of the query defining it) may trigger the execution of new calls (perhaps even to the same service), etc. While some form of recursion is useful, e.g., for defining transitive closure type of computations, detecting termination in general is difficult due to the distributed form of the computation and the independence of the peers.



## 4 Data and Computation Model

In this section, we briefly define the AXML data model and the semantics of AXML documents and services. Our presentation here is informal. The formal definitions as well as the proofs of the results can be found in [2].

Intuitively, an AXML instance consists of a number of peers, each one containing some AXML documents that are being run. AXML documents are XML unordered trees. The evaluation of these documents generates calls between these peers and possibly results in new documents being evaluated at each peer. As we shall see, the evaluation is nondeterministic. This captures the asynchronous evolution of the global instance, which may eventually reach a fixpoint or not. We will first present the data model, then the computation.

### 4.1 Data Model

An (AXML) instance consists of a number of peers. Each peer contains AXML documents, some service definitions and a working area. We next define instances, then proceed to the definition of documents and services.

**Instances.** An *instance*  $\mathcal{I}$  consists of a number of peers  $p_1, \dots, p_n$ . The *content* of a peer  $p_i \in \mathcal{I}$  is defined by a triple  $(\mathcal{R}_i, \mathcal{F}_i, \mathcal{W}_i)$ , where  $\mathcal{R}_i$ , the peer's *repository*, is a set of persistent AXML documents,  $\mathcal{F}_i$ , the peer's *services*, is a set of AXML service definitions, and  $\mathcal{W}_i$ , the peer *working area*, is a set of AXML temporary documents. All the sets are assumed to be finite.

Each document  $d$  in the working area  $\mathcal{W}_i$  of a peer  $p_i$  represents the computation of some service call in  $p_i$ , i.e., some current work that  $p_i$  is performing. Any such document  $d$  also contains a *destination* attribute specifying the place where the result of this computation should be sent, which can be a local element or a request from a remote peer.

**Documents.** As for standard XML documents, an AXML document is modeled by a labeled tree with nodes representing the document elements or attributes and with edges capturing the component-of relationship among document items. The three main differences from the standard XML data model [50] are that (1) we ignore here the order of elements, hence our trees are unordered,<sup>9</sup> so we only consider the order-free fragments of XPath (for parameters) and XQuery (for service definitions); (2) a validity predicate is attached to some elements to specify when some particular data become stale; (3) some of the tree leaves are special service call nodes, called in the following *sc-nodes*. An sc-node is labeled by a tuple of the form  $\langle p, f, x_1, \dots, x_n \rangle$  where:

- $p$  and  $f$  are respectively a *peer* and *service* names, or XPath expressions. In the first case, the service  $f$  must be defined in peer  $p$  with arity  $n$ .
- $x_1, \dots, x_n$ , the call parameters, are AXML documents or XPath expressions.

<sup>9</sup> We may take into account the ordering in some specific cases, e.g., for the extensional portions of documents.

An sc-node where none of  $p, f, x_1, \dots, x_n$  are XPath expressions is called a *concrete* call.

**Reduced documents.** Continuous services send a sequence of answers to the caller. Smart (or optimized) services may only send the delta since the last answer. In other cases, the caller may be responsible for detecting and ignoring redundant data. To abstract this (without having to get into implementation details such as who performs the optimization and when), we use in the formal model the notion of *reduced* version of a document, where multiple occurrences of the same data are omitted.

To define reduced documents, we use the auxiliary concept of *inclusion relationship* among trees. A reduced document is such that no subtree is included in one of its siblings. One can show that the reduced version of a document is unique. It can be, for instance, computed by iteratively removing redundant subtrees. We will assume in the sequel that all our AXML documents are reduced.

**Service definitions.** To conclude this section, let us consider the definition of  $\mathcal{F}_i$ , i.e., the definition of services. The semantics of XQuery queries is standard, with one notable exception: when evaluating path expressions, service calls act like document boundaries that the evaluation cannot cross. In other words, they are terminal nodes that do not match any path expression.

The definition of an AXML service consists of the service name; the service type (e.g., continuous or not); the service parameter names  $v_1, \dots, v_n$ ; and a parametrized query  $Q(v_1, \dots, v_n)$ , namely a query that may refer to the (parameters) documents  $v_1, \dots, v_n$ .

## 4.2 Computation

We are now ready to define the semantics of AXML documents. Each peer includes a collection of AXML trees in  $\mathcal{R}_i$  and  $\mathcal{W}_i$ . These documents may contain service calls that may be activated to derive more information about the documents. A service call activation spans a computation on one of the peers. More precisely, the activation in peer  $s$  of a particular service call to peer  $r$  involves (1) (possibly) instantiating in  $s$  the XPath expressions of attributes of the call, (2) for each instantiation, sending concrete calls from peer  $s$  to peer  $r$ , (3) computing in peer  $r$  the corresponding answers and (4) returning the answers to peer  $s$ , where they are received and merged at the appropriate place in the tree. If, for some reason, the resulting tree is no longer a legal AXML document, it becomes the *inconsistent* document.

Recall that the decision whether service calls can, and need to, be instantiated (resp. sent, computed, returned) at a given time depends on the specific call attributes. We will simply refer to such calls as *eligible for instantiation* (resp. *sending*, *computation*, *returning*). We will see in the next section how this can be implemented.

An *initial instance* is such that all peers have an empty working area. Given an initial instance  $\mathcal{I}$ , each peer  $s = \langle \mathcal{R}, \mathcal{F}, \mathcal{W} \rangle$  evolves in a similar way. Starting from  $\mathcal{I}$ , repeatedly (and nondeterministically) one of the following steps is executed:

**Step 1: Instantiate the XPath parameters.** For some (nonconcrete) sc-node  $v$  in  $\mathcal{R}$  or  $\mathcal{W}$  that is eligible for instantiation, the XPaths are evaluated, and for each instantiation, a new document is added to  $s$ 's working area. The roots of these documents have the

corresponding concrete service call as an *sc-node* child, and have  $v$  as the destination for the result of the computation.

**Step 2: Send/Receive an external call.** For some concrete *sc-node*  $n$  in  $\mathcal{R}$  or  $\mathcal{W}$  that is labeled with a call  $c$  to some remote peer  $r$  and is eligible for sending, the call is activated. Formally, this consists in adding, to the working area of the receiving peer  $r$ , a new document whose root has an *sc-node* child labeled with  $c$  and having  $n$  as the destination for the result.

**Step 3: Compute a local call.** For some concrete *sc-node*  $n$  in  $\mathcal{R}$  or  $\mathcal{W}$  that is labeled with a call  $c$  to a local service of  $s$  and is eligible for computation, evaluate the service query using the given parameters. The result, a forest, is merged under the parent node of  $n$ .

**Step 4: Return/Receive result of a call.** For some document  $d$  in  $\mathcal{W}$ , eligible for being returned as an answer, the children of  $root(d)$ , (not including  $d$ 's *destination* attribute) are sent to the destination peer and merged under the parent of the destination node.

Observe that, in the above computation, we grouped *sending* (resp. *returning*) a call and *receiving* it in one operation. Intuitively, our send/receive (resp. return/receive) operation captures the moment when the receiver receives the message. Finally, to guarantee a correct semantics, we need some *fairness* condition: *Any operation that may happen, eventually happens.*

**Nondeterminism and confluence.** In general, an initial instance  $\mathcal{I}$  may be transformed in many different ways, depending on the choice of the operations to perform. This nondeterminism is built in the semantics. So, even if an instance converges to a fixpoint, the fixpoint does not have to be unique. Furthermore, as mentioned in Sect. 3.6, the computation may continue forever, building more and more data, i.e., there is no guarantee of termination.

Although this may seem to be a negative feature of the model, observe that this naturally models the real world we are trying to capture. The state of a peer may continuously evolve because, for instance, of interactions with human users updating data. Also, continuous external services such as subscriptions may keep sending new data to the peer. So, the system should not be expected to terminate. Also, data may expire or get deleted and the order in which the various operations/queries are executed may have impacts on the state. Thus, because of the asynchronicity and the independence of peers, determinism is an elusive goal in such an environment. However, termination and confluence can be enforced under very strict restrictions, as outlined next.

*Remark 2. (Monotone computation.)* Suppose the computation is monotone, i.e., no fact is ever deleted or updated, and information keeps being added in a cumulative manner. Furthermore, assume that each call becomes eligible for instantiation and sending infinitely often (i.e., the call is repeatedly triggered and its XPath re-instantiated). Under these restrictions, the order in which the steps are executed is no longer significant. One can then guarantee that all computations lead to a (possibly infinite) unique state. A finite state can be guaranteed with some additional restrictions. This is in the spirit of results on inflationary fixpoint semantics, see [5].

## 5 Limiting the Firing of Calls

So far, we have described the AXML paradigm at a rather abstract level. Before we consider its actual implementation, we highlight some important issues that are critical for a real-life implementation. Before activating a service call, two points need to be checked: that the receiving peer is willing to accept the call, i.e., that the caller has the proper privileges to issue the call, and that the receiver has the capability to process the call, which involves understanding the parameters that are sent. In practice, access to services from other peers will be severely controlled for security reasons. Also, peers will have limited capabilities, e.g., most of them will only accept calls with 'plain' XML arguments.

### 5.1 Site Capabilities and Security

First, consider peer capabilities. We illustrated in Sect. 3.1 the use of intentional parameters in a service call. Observe that they may, in principle, be evaluated *before* or *after* the call is sent to the receiving peer. In practice, not all choices are always feasible. For instance, consider again the example in Sect. 3.1. If `auction.com` is not capable of calling `getBudget` on `bank.com` (e.g., because it is not an AXML peer), then 'Bob's AXML peer must first call `getBudget`, and only then call `getOffers` with the result.

Now, consider the security concerns that must guide call activations. Access control is a needed feature for many applications. First, a service provider may wish to reserve the access to a service to those who paid for it. For example, `acm.org` currently allows users from the `inria.fr` domain to use the search services of the ACM digital library, but not any Web user can do so. Furthermore, security is necessary to protect sites from a malicious usage. Not surprisingly, the exchange of data that includes service calls is a major security hole. For instance, suppose that we want to break into a peer *s*, say the site `god.com`, providing quotes of the day. There are two main ways to do this:

**In a call parameter.** Intentional service parameters open backdoors to AXML servers. For instance, a malicious client may use the following call to `god`:

```
<sc>god.com/QuoteOfDay
  (<sc>buy.com/BuyCar ("BMW") </sc>)
</sc>
```

This malicious user does not wish to buy the car by himself, but tries to make `god.com` buy it.

**In a call result (Trojan Horse).** Suppose now that `god.com` is malicious in the quotes it provides, e.g., by returning the following quote as a call result:

```
<quote> Love means never having to say you're sorry.
<sc>buy.com/BuyCar ("BMW") </sc></quote>
```

Thus, by sending an intentional result, the *god* peer may force its clients to invoke dangerous services.

Finally, perhaps the most natural violation of security is to bring an AXML peer to transmit private data to a malicious distant site. This may be achieved, for instance, by including the following call (as a parameter of a call or in a result):

```
<sc>i.am.bad/SneakAbout ([..../{*}])</sc>
```

Instantiating this XPath argument amounts to sending *i.am.bad* (possibly private) parts of the document that included this call, which is clearly an issue.

The above examples show that the AXML framework makes unauthorized attempts to access data quite likely, as well as malicious usage of Web services. Hence, access control is essential. We next see how this can be incorporated in the framework.

## 5.2 Our Solution

We illustrate how the above issues may be addressed with two very simple policies. These policies have to be combined with some access control mechanism on the documents. Access models for XML have been proposed in, for example, [18] and aspect will not be detailed here.

In the first policy, called *binding*, a peer publishes the kind of arguments each of its services accepts (e.g., arbitrary AXML, XPath expressions, strict XML). Only calls with the proper arguments may then be activated. Note that this policy can be enforced using the WSDL language, which enables publication of XML Schema types for services input/output parameters. We proposed in [33] a set of algorithms that follow this approach.

The second policy, called *trust*, reflects some form of agreement between the caller and the receiver. More precisely, the reasoning that allows one to decide whether a service *sv* (where *sv* includes the name of the service and the site that provides it) can be called by a site *S* is encapsulated in a boolean function *canCall*(*sv*, *S*). The *canCall*(*sv*, *S*) function returns *true* if *S* is willing to call *sv* and the provider of *sv* is willing to accept this call from *S*. Note that, like in Java's sandbox security model [22], the decision depends on the origin of the call. This function will be used to determine which calls are eligible for activation at each point in time. We will see exactly how this is done in the next section.

To implement *canCall*, we can assume, for instance, that each peer has an *agreed service list*, containing the services that it trusts and is willing to call. Similarly, we assume for every service, an *agreed site list*, i.e., the sites (trusted and accepted by the service provider) from which the service may be called. These two lists are typically exposed as Web services. More precisely, each AXML peer *S* provides a service to check whether it is willing to let another peer *S'* call one of its services and a service to check whether it is willing to call some particular service. For non-AXML peers, we make conservative assumptions.

As mentioned above, these two models, *binding* and *trust*, may be combined. They may also be extended in a number of ways. First, one may want to include

some access control list (ACL) mechanism, to grant different rights to various users of a peer. One might want to control the right to fire a particular service call or the right to access data with an arbitrary granularity (e.g., at the element level). Also, the *canCall* function may vary in time. For instance, depending on the load of the service provider, one may want to restrict usage of the service to certain clients only. Finally, one may want to include arbitrarily complex solutions for trust management that have been proposed such as REFEREE [16]. No matter how complex the used policy is, the provider essentially needs to know, given a concrete call and a site, whether this site is entitled to activate this particular call.

## 6 Evaluation and Implementation

In this section we describe the architectural components and the algorithms used by an AXML peer in order to evaluate and maintain AXML documents. First, we explain how time-related events are detected in the system. Then, we see how the evaluation of documents is affected by these events.

**The Event Detector.** To capture time-related events, we use an *Event Detector* module (ED). For simplicity, we omitted this module from the architecture sketch (Fig. 1) at the beginning of this chapter. The ED of an AXML peer  $P$  monitors all AXML documents on  $P$ , including data validity parameters, and the activation mode and frequency of all service calls present in these documents. The ED sends messages to other components of the AXML peer:

- to the Evaluator: when a service call has expired, or has reached timeout
- to the AXML storage: when a data node has become invalid

Before presenting our evaluation algorithm, recall from Sect. 3.4 that service calls can be defined to be *immediate* or *lazy*. Immediate service calls have to be activated as soon as they expire, while the activation of expired lazy calls may be postponed until their results are actually needed. To simplify the presentation, we first assume below that all the service calls in the documents are defined with an *immediate* execution mode, and explain the evaluation algorithm for this restricted case. Then, we explain how the above needs to be extended in order to support *lazy* calls. Finally, we describe our implementation. Recall from Sect. 4 that a *concrete* service call is one whose parameters do not include XPath expressions.

### 6.1 Calls with Immediate Mode

We start by explaining how the Evaluator decides when a call is *eligible* for instantiation, resp. activation, computation and return (in the terms of Sect. 4), based on the messages received from the ED. We then outline the algorithms for processing service call activations.

**Deciding on call eligibility.** The following rules are applied by the Evaluator module:

- Upon receiving an “ $sc$  has expired” message from the ED, if  $sc$  is nonconcrete, it becomes *eligible for instantiation*.
- If  $sc$  is concrete and aimed at some service outside  $P$ , we first choose some of the service calls included in the parameters (according to the security, capability and optimization reasoning outlined in Sect. 5), and process them. Only then does  $sc$  become *eligible for sending*.
- If  $sc$  is concrete and aimed at a local AXML service defined on  $P$  by a query  $Q$ , then  $sc$  becomes *eligible for computation*.
- After an  $sc$  aimed at a local AXML service was evaluated, its result becomes *eligible for returning* after being postprocessed (again, by calling some of the service calls in the result, based on security, capability, etc.).

**Processing service call activations.** Recall from Sect. 4 that the four steps of computation were chosen nondeterministically and in random order. We introduce here the notion of *task*, to track the evaluation of each particular service call, from the moment it is activated to the end of its evaluation. Like  $sc$ -nodes, tasks can be *concrete* or *nonconcrete*. Documents in  $\mathcal{W}$  naturally have corresponding tasks, and so do activated  $sc$ -nodes in  $\mathcal{R}$ . Note that the evaluation is still nondeterministic, and that tasks can be evaluated in parallel: at a given point in time, a task is either *running*, *ready* or *suspended*, waiting for some event, perhaps the end of another task. Any of the ready tasks may be processed at that point.

Tasks are created in three possible ways. First, the Evaluator creates a new task (concrete or nonconcrete) for the activation of every expired immediate service call. Second, upon receiving from outside a call to a service defined at  $P$ , the SOAP wrapper creates a task for this call in  $\mathcal{W}$ . Note that this task is concrete, since only concrete tasks can be sent (Sect. 4). Third, the processing of a task may create other tasks, as we will see.

As a notation, let  $t(d, P_f, f, p_1, p_2, \dots, p_n)$  be a task with destination  $d$ , corresponding to the activation of a call to the service  $f$ , provided by the peer  $P_f$ , with parameters  $p_1, p_2, \dots, p_n$ .

Figure 2 outlines the simple algorithm for evaluating nonconcrete tasks. First, the XPath parameters of the task have to be evaluated, by issuing calls to the query processor. When the evaluation is done, each  $p_i$  has the value of an AXML forest  $f_i$ . As mentioned in Sect. 3.1, the nonconcrete call is unrolled into as many concrete calls as there are elements in the Cartesian product of the forests  $f_i$ . The processing of  $t$  is over when all these concrete tasks have finished.

In Fig. 3, we describe the processing of a concrete task. Assume that a parameter  $p_i$ , which is an AXML tree, contains some expired service call  $sc$ . Then,  $P$  has to decide whether it needs to activate it or to send it as an intentional parameter. This decision is based on the *binding* and *trust* policies described in Sect. 5.<sup>10</sup> Note that the decision is made locally, using the policies of  $P, P_f, sc$  without requiring a ‘global’ view of the security and capability requirements of other peers.

At line 6, if  $f$  is a service local to  $P$ , then we call the XQuery processor with the proper arguments; otherwise, a call is sent to  $P_f$  via the SOAP wrapper. In both

<sup>10</sup> It may also take into account other considerations such as the system load.

	peer $P$ , nonconcrete task $t(d, P_f, f, p_1, p_2, \dots, p_n)$
1	evaluate the XPath parameters $p_1, p_2, \dots, p_n$
2	foreach $i = 1, 2, \dots, n$
3	let $f_i$ be the value obtained for $x_i$ (an AXML forest)
4	foreach $x = (x_1, x_2, \dots, x_n) \in f_1 \times f_2 \times \dots \times f_n$
5	create $t_x(P_f, f, x_1, x_2, \dots, x_n, t.root)$
6	insert $t_x$ in $\mathcal{W}$
7	suspend until all $t_x$ finish
8	exit

Fig. 2. Processing a nonconcrete task

cases,  $t$  is suspended waiting for the result. Once  $P$  receives the result, if it needs to forward it to the distinct peer  $d.peer$ , we may have to decide when and where to execute the calls it contains. The reasoning is very similar to the one above, dividing the work among  $d.peer$  and  $P$ . Subsequently, the result is sent to  $d$ . (If  $d$  is local, by accessing the local AXML repository; otherwise, by sending a result message through the SOAP wrapper.) Finally, the concrete tasks exits.

**Continuous tasks.** Tasks associated with calls to continuous services keep running. The received updates just keep being sent to their destination. When they appear in the algorithms described above, these tasks are nonblocking.

**Unsubscribe and timeout.** For readability, we have omitted some issues from the algorithms depicted in Figs. 2 and 3. First, if an unsubscribe message for a continuous service is sent by the ED, the Evaluator identifies the associated concrete tasks, sends “unsubscribe” messages to their service providers and destroys the task. Similarly, when a noncontinuous call times out, the Evaluator destroys its task.

## 6.2 Calls with Lazy Mode

Let us now consider the more complex case of the lazy mode.

**Service call dependencies.** The presence of lazy calls may cause dependencies among call activations. For example, assume that we need to activate a nonconcrete service call. Before instantiating its XPath parameters, we may need to activate some lazy service calls, which may affect the result of the instantiation. This situation is illustrated in the AXML document shown in Fig. 4(a). The ‘influence zone’ of  $sc_2$ , i.e., the set of nodes that may be modified by the results of  $sc_2$ , intersects the zone in which the XPath parameters of  $sc_1$  are evaluated. If  $sc_2$  is in lazy mode and has expired, then it is preferable to call it again before we instantiate the XPath arguments of  $sc_1$ . In turn,  $sc_2$  may have XPath parameters that evaluate in the influence zones of lazy expired service calls, leading to a graph of dependencies like that in Fig. 4(b).

Similarly, assume that a request for an AXML service is received and the service query  $Q$  needs to be evaluated. Before calling the XQuery processor, we have to check if the data read by  $Q$  intersects the influence zone of some lazy expired service call. This again leads to a dependency graph of the above form.



	peer $P$ , concrete task $t(P_f, f, p_1, p_2, \dots, p_n, d)$
1	foreach $sc_i$ in $p_1, \dots, p_n$
2	if $P$ decides to activate $sc_i$
3	then create $t_i$ , new task for $sc_i$ ; insert $t_i$ in $\mathcal{W}$
4	suspend until all $t_i$ finish
5	if $P = P_f$ (i.e., $f$ is defined in $P$ by some query $Q$ )
6	then call $Q(p_1, p_2, \dots, p_n)$ ; suspend until result ready
7	else (i.e., $f$ is a distant service)
8	call $P_f / f(p_1, p_2, \dots, p_n)$ ; suspend until result ready
9	if $P \neq d.peer$
10	then foreach $sc_j$ in result
11	if $P$ decides to activate $sc_j$
12	then create $t_j$ , new task for $sc_j$ ; insert $t_j$ in $\mathcal{W}$
13	suspend until all $t_j$ finish
14	send result to be inserted under $d$
15	if $f$ noncontinuous
16	then exit

Fig. 3. Processing a concrete task

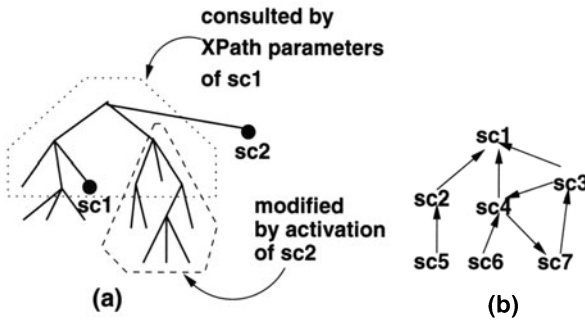


Fig. 4. Dependencies among service calls

A reasonable compromise between precision and complexity has to be found for tracking dependencies. It is not possible to compute dependency graphs statically. For instance, as a document evolves, calls are added, or removed, by service call activations. Computing the *exact* dependency graph of a service call leads to computationally complex problems such as XPath containment [20].

We therefore adopt the following pragmatic solution. We consider the influence zone of a service call to be all the subtrees rooted at its parent. We consider the scope of an XPath expression to be the set of subtrees rooted in the highest nodes attained by its evaluation, as described by the XPath specification [52]. Finally, we assume the data read by an XQuery query to be described by the XPath expressions in its `for` clause.<sup>11</sup> In general, path expressions may also appear in other parts of the query,

<sup>11</sup> In some sense, this simple approach is pessimistic, since we do not use the `where` clause to filter the actually consulted data.

e.g., the *where* clause. Without loss of generality we assume here that the query is first normalized [31]. We have thus brought the dependency decision problem to deciding whether two trees intersect, which can be done in constant time, provided a convenient encoding for element IDs (e.g., [30]).

A call dependency graph may contain cycles reflecting mutual call dependencies. They are broken by arbitrarily choosing some dependencies to be ignored. Breaking the cycles amounts to introducing nondeterminism and possibly ‘missing’ some data. In a Web context, this is acceptable.

**Eligibility with lazy mode.** In the presence of lazy calls, a given call *sc* may be declared eligible for instantiation (resp. execution) only after all the lazy calls in its data dependency graph have been issued.

**Call activation with lazy mode.** Task processing in the presence of lazy calls is more complex due to the fact that we have to track data dependencies. First, before instantiating an XPath argument of a nonconcrete call, we have to make sure that the data it bears on is available. To that purpose, before line 1 in Fig. 2, we need to construct the dependency graph *G* for the XPath parameters of the task, on a snapshot of the destination document. If *G* has cycles, they are broken; then, we create tasks for all the leaf nodes from *G* and process them in parallel. When these tasks are over, to take into account their effect on the destination document, we recompute *G*. As long as *G* is not empty, we repeatedly create and process tasks, corresponding to lazy expired calls, that *t* depends on. The processing of *t* is suspended until *G* is empty.

The very same steps have to be applied when processing a concrete task, before actually calling the XQuery processor (line 6 in Fig. 3), except that *G* is computed for the XPath expressions that *Q* depends on. We omit the details.

### 6.3 Implementation

A first prototype of AXML peer software has been implemented in Java. It relies on the XOQL query processor [7], which implements an algebra similar to the one of XQuery.<sup>12</sup> The SOAP wrapper, which is needed both to invoke and answer service calls, relies on the Axis engine from the Apache software group [9], which although in early development stage, provides good performance and great flexibility through its architecture based on chainable handlers.

We implemented the evaluation strategy of Sect. 6.1, which only deals with the immediate activation of service calls. This is done mainly using a timer thread that acts as a scheduler. In this restricted case, dependency among service calls does not have to be tracked. *Tasks* are evaluated in parallel, each one being handled by a separate thread. A thread pool mechanism is used to limit the number of simultaneously running threads.

Since SOAP supports only RPC calls and one-way messages, we built a layer on top of it to allow for continuous services [17]. Basically, the caller of a continuous service provides a listening SOAP service, used by the callee to return a stream of

<sup>12</sup> We chose XOQL because at the time we started this implementation, no XQuery processor was available to us. Although there are differences from XQuery, these are mostly syntactic.

answers as one-way messages. This prototype is functional, and was used to build a distributed peer-to-peer auctioning application, where each peer can offer auctions for other peers to bid on, and search for auctions of interest available from other peers, without needing a centralized auction server [3].

## 7 Conclusion

The AXML paradigm allows us to turn service calls embedded in XML documents into a powerful tool for data integration. This includes, in particular, support for various integration scenarios like mediation, data warehousing and distributing computations over the Web via the exchange of AXML documents.

We implemented a first prototype, but further work is needed to develop appropriate optimization techniques. Because of the richness of the model, this is a complex task that should borrow from many techniques that have been previously used, notably in the contexts of warehouses and mediators. We also need to build an environment for designing AXML documents and tools for easily building applications that use them.

In [33], we proposed to control the exchange of AXML documents between applications by using schemas for the input and output parameters of Web services. We developed a set of novel algorithms to make an AXML document match a given exchange schema, by invoking some of the service calls it contains.

We also extended AXML to deal with documents that are distributed and/or replicated among several peers [4]. We developed an associated cost model for query evaluation, and an algorithm to find an optimal strategy of replication for a given peer.

The proposed AXML paradigm should be further experimented and evaluated. Toward this goal, we are intending to use AXML as an application development platform in the context of a European project called DBGlobe. The project deals with data management problems in global distributed computing environments, with a strong emphasis on mobility. We believe it provides an adequate testbed for the proposed framework.

### 7.1 Further topics

As we saw in Sect. 5, security is an important topic in the context of AXML, because of the exchange of data containing service calls. Security protocols currently being developed for XML and Web services by standards organizations should be extended to handle the exchange of AXML data among peers.

As mentioned earlier, developing further optimization techniques is also an important issue for AXML. For instance, query evaluation on AXML data can be distributed among several peers, and new kinds of (possibly distributed) indexes can be developed with this idea in mind.

Another important issue is the role played by service directories, in the style of UDDI. AXML can serve as a basis to develop distributed directories, by contrast with

the existing centralized ones. Also, richer results can be returned by directories if their answers consist of AXML data.

We have conducted some research on the theoretical foundations of the AXML model, but clearly more investigation is needed. The semantics can be refined to determine a certain number of properties of the model, such as a better characterization of termination conditions, or soundness/completeness of lazy query evaluation on AXML documents.

## References

1. S. Abiteboul, B. Amann, S. Cluet, A. Eyal, L. Mignet, and T. Milo. Active views for electronic commerce. In *Proc. of VLDB*, 1999.
2. S. Abiteboul, O. Benjelloun, and T. Milo. A data-centric perspective on web services (preliminary report). Technical Report 212, INRIA, November 2001.
3. S. Abiteboul, O. Benjelloun, T. Milo, I. Manolescu, and R. Weber. Active XML: Peer-to-peer data and web services integration (demo). In *Proc. of VLDB*, 2002.
4. S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. In *Proc. of ACM SIGMOD*, 2003.
5. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, Reading, MA, 1995.
6. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *Int. Journal on Digital Libraries*, 1(1):68–88, April 1997.
7. V. Aguilera. The X-OQL home page.  
<http://www-rocq.inria.fr/~aguilera/xoql>.
8. N. Alon, T. Milo, F. Neven, D. Suciu, and V. XML with data values: typechecking revisited. In *Proc. of ACM PODS*, 2001.
9. The Apache Software Foundation. <http://www.apache.org>.
10. A. Bonifati, D. Braga, A. Campi, and S. Ceri. Active XQuery. In *Proc. of ICDE*, 2002.
11. A. Bonifati, S. Ceri, and S. Paraboschi. Pushing reactive services to xml repositories using active rules. In *Proc. of the Int. WWW Conf.*, Hong Kong, China, May 2001.
12. L. Cardelli. Abstractions for mobile computation. In *Secure Internet Programming*, pages 51–94, 1999.
13. L. Cardelli and A. D. Gordon. Mobile Ambients. In M. Nivat, editor, *Proc. of FoSSaCS*, volume 1378, pages 140–155. Springer, Berlin Heidelberg New York, Berlin, Germany, 1998.
14. R. G. G. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Mateo, CA, 1994.
15. V. Christophides, R. Hull, A. Kumar, and J. Siméon. Workflow mediation using VorteXML. *IEEE Data Engineering Bulletin*, 24(1):40–45, March 2001.
16. Y. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: trust management for Web applications. In *Proc. of the Int. WWW Conf.*, volume 29(8-13), pages 953–964, 1997.
17. F. Cremenescu. Supporting Subscription Services using SOAP, 2001. Stage de fin d’étude, Ecole Polytechnique.
18. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Securing XML documents. In *Proc. of EDBT*, 2001.
19. A. Deutsch, M.F. Fernandez, D. Florescu, A.Y. Levy, and D. Suciu. A query language for XML. In *Proc. of the Int. WWW Conf.*, volume 31(11-16), 1999.

20. A. Deutsch and V. Tannen. Containment of regular path expressions under integrity constraints. In *Proc. of the KRDB Workshop*, Rome, 2001.
21. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS approach to mediation: data models and languages. *Journal of Intelligent Information Systems*, 8:117–132, 1997.
22. L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers. Going beyond the sandbox: an overview of the new security architecture in the Java Development Kit 1.2. In *Proc. of the Usenix Symp. on Internet Technologies and Systems*, 1997.
23. A. Gupta. *Integration of Information Systems: Bridging Heterogeneous Databases*. IEEE Press, 1989.
24. R. Halstead. Multilisp: A language for concurrent symbolic computation. *ACM Trans. on Programming Languages and Systems*, 7(4):510–538, 1985.
25. H. Hosoya and B. C. Pierce. XDuce: A typed XML processing language (preliminary report). In *Proc. of WebDB*, May 2000.
26. T. Jim and D. Suciu. Dynamically distributed query evaluation. In *Proc. of ACM PODS*, pages 413–424, 2001.
27. The Kazaa home page. <http://www.kazaa.com>.
28. T. Lahiri, S. Abiteboul, and J. Widom. Ozone: Integrating structured and semistructured data. In *Proc. Int. Workshop on Database Programming Languages*, 1999.
29. A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, pages 251–262, 1996.
30. Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *Proc. of VLDB*, 2001.
31. I. Manolescu, D. Florescu, and D. Kossmann. Answering XML queries over heterogeneous data sources. In *Proc. of VLDB*, 2001.
32. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. Technical report, Stanford University Database Group, Feb 1997.
33. T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, and F. Dang Ngoc. Exchanging intensional XML data. In *Proc. of ACM SIGMOD*, 2003.
34. The Morpheus home page. <http://www.morpheus-os.com>.
35. B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda. Monitoring XML data on the Web. In *Proc. of ACM SIGMOD*, 2001.
36. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems, 2nd Edition*. Prentice-Hall, 1999.
37. Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proc. of VLDB*, pages 413–424, 1996.
38. J. Powell and T. Maxwell. Integrating Office XP Smart Tags with the Microsoft .NET platform. <http://msdn.microsoft.com>, 2001.
39. Simple Object Access Protocol (SOAP) 1.1. <http://www.w3.org/TR/SOAP>.
40. I. Tatarinov, Z. Ives, A. Levy, and D. Weld. Updating XML. In *Proc. of ACM SIGMOD*, 2001.
41. Universal Description, Discovery, and Integration of Business for the Web (UDDI). <http://www.uddi.org>.
42. J.D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.
43. The World Wide Web Consortium (W3C). <http://www.w3.org>.
44. G. Weikum, editor. *Infrastructure for Advanced E-Services*, volume 24, no. 1. Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society edition, March 2001.

45. J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.
46. G. Wiederhold. Intelligent integration of information. In *Proc. of ACM SIGMOD*, pages 434–437, Washington, DC, May 1993.
47. Web Services Definition Language (WSDL). <http://www.w3.org/TR/wsdl>.
48. Web Services Flow Language (WSFL 1.0).  
Available from <http://www.ibm.com/>.
49. XLANG, Web Services for Business Process Design.  
[http://www.gotdotnet.com/team/xml\\_wsspecs/xlang-c](http://www.gotdotnet.com/team/xml_wsspecs/xlang-c).
50. XML 1.0 (2nd edition). <http://www.w3.org/TR/REC-xml>.
51. XML Schema. <http://www.w3.org/TR/XML/Schema>.
52. XML Path Language (XPath) Version 1.0. <http://www.w3.org/TR/xpath>.
53. XQuery 1.0: An XML Query Language. <http://www.w3.org/TR/xquery>.

---

# WebVigiL: An Approach to Just-In-Time Information Propagation in Large Network-Centric Environments

Jyoti Jacob, Anoop Sanka, Naveen Pandrangi, and Sharma Chakravarthy

The University of Texas at Arlington, Arlington, TX 76019, USA  
{jacob, asanka, pandrang, sharma}@cse.uta.edu

**Summary.** In this chapter, we argue that push technology is critical for a large number of applications – traditional as well as network-centric. Push technology can be implemented in various ways depending upon the ‘openness’ of the underlying systems into which the push technology is being inserted. We have experimented with several approaches: integrated, mediated (using intelligent agents) and wrapper-based. Intelligent agents are useful for supporting push technology in situations where the underlying system/software provides some hooks but is not open to user modification. In this chapter, we first overview the need for push and pull technologies. We then provide an overview of active capability for detecting changes in databases and other structured environments.

In addition to traditional database applications, a number of newer applications can benefit from the concepts that came out of active capability. For example, efficient and effective change detection and notification of Web pages is becoming increasingly important for environments such as the Web and distributed heterogeneous systems. The WebVigiL project’s objectives are to investigate the specification, management, and propagation of changes as requested by a user in a timely manner while meeting the quality of service requirements. In this chapter, we elaborate on the issues that need to be addressed, and on our preliminary approach. We present an architecture and discuss the functionality that needs to be supported by various modules in the architecture. We use the active capability in the form of event–condition–action (ECA) rules and a combination of push/pull paradigms for this problem.

## 1 Introduction

Active rules have been proposed as a paradigm to satisfy the needs of many database and other applications that require a timely response to situations. Event–condition–action (ECA) rules are used to capture the active capability in a system. The utility and functionality of active capability (ECA rules) has been well established in the context of databases. In order for the active capability to be useful for a large class of advanced applications, it is necessary to go beyond what has been proposed/developed in the context of databases.

There are a number of situations where one needs to know when changes are made to one or more documents that are stored in a distributed (typically heterogeneous)

environment. The number of documents that need to be monitored for changes is large, and they are spread over multiple information repositories. The emphasis here is on selective notification; that is, changes are notified to appropriate persons/groups based upon interest (or profile/policy) that has been established earlier. Also, there should be a mechanism for establishing the interests/profiles/policies themselves. Typically, change detection is done either manually or by using queries (pulling information by polling) to check whether any document of interest has changed since the last check. This entails wasted resources and at the same time does not meet the timeliness of change detection and associated notification.

As an example, the above situation is very common in large software development projects involving a large number of documents, such as requirement analysis, design specification, detailed design document, and implementation documents. The life cycles of such projects are in years (and some in decades), and changes to various documents of the project take place throughout the life cycle. Typically, a large number of people are working on the project, and managers need to be aware of the changes to any of the documents to make sure the changes are propagated properly to other relevant documents and appropriate actions are taken. Moreover, large software developments take place in distributed environments. Information retrieval in the context of the Web is another example that has similar characteristics. Different users may be interested in knowing changes to specific Web pages (or even combinations thereof), and want to know when those changes take place. Some examples are: students want to know when the Web contents of the courses they have registered for change; users may want to know when news items are posted with some specific context in which they are interested. In general, the ability to specify changes to arbitrary documents and get notified in different ways will be useful for reducing the wasteful navigation of Web in this information age. The approach proposed in this chapter will free the user from having to constantly monitor for changes using the pull paradigm. In addition, this approach also provides a powerful way to disseminate information efficiently without sending unnecessary or irrelevant information.

We believe that some of the techniques developed for active databases, when extended appropriately, along with new research extensions will provide a solution to the above class of problems. In addition, there is the theoretical foundation for event specification and its detection in centralized and distributed environments. The main objective of this effort is to develop the theory, architecture, and prototype implementation of a selective propagation approach that can be applied to Web and other large-scale network-centric environments. We will draw upon the techniques developed for Sentinel (an object-oriented active DBMS) and reexamine them from a broader, general-purpose context.

The remainder of the chapter is organized as follows. In Sect. 2, we give an overview of related work. In Sect. 3, we discuss the push/pull paradigms and their relevance to the change detection problem. In Sect. 4, we present architecture and discuss the functionality of the components. Finally, we discuss future work and draw some conclusions in Sect. 5.



## 2 Related Work

Considerable research has been done in active database [4, 12, 23]. In WebVigiL, the active capability is used for the run-time management of sentinels, as without an event-based approach, asynchronous monitoring is not possible. Several tools have been developed and are currently available for some of the problems addressed in WebVigiL. AIDE, developed by AT & T [13] displays the difference between two HTML pages. Changedetection.com [8] allows users to register their request and notifies them when there is a change. The change is always detected at the page level and not at the granularity chosen by the user, as in WebVigiL. WebCQ [18] supports customized change detection and notification. It tracks changes to a finer level of granularity in a page. However, changes are detected only between subsequent versions of the page, unlike WebVigiL, which provides flexibility in specifying the reference page. In Xyleme [9, 19, 24], the idea of active paradigm is being used for detecting changes by evaluation of continuous or monitoring queries on XML/HTML documents. The focus is on the subscription language and continuous queries. Lifespan and dependent sentinels are not supported. Change detection algorithms are simpler in WebVigiL because changes are detected on the information seen by the user (i.e., on the contents and not on the structure). Finally, unlike Xyleme, WebVigiL supports both user-specified frequency and on change (for page fetches). WebVigiL offers user-centric specifications in terms of on change (for fetching) and best effort (for notification) that are important from users' viewpoint. The adaptive push/pull [11] approach evaluates the effect of various approaches (push, pull, and combinations thereof) from the point of view of propagating changes from server to the client. WebVigiL addresses a different problem – that of change detection based on an expressive user specification. Finally, the use of ECA rules and the optimisations that accompany its usage are novel to WebVigiL.

## 3 Push/Pull Paradigms

The traditional approach to information management has been through the use of a database management system (DBMS). Early DBMSs were developed to satisfy the needs of certain classes of business applications (mainly the airline and banking industries). The requirements of these industries were to store, retrieve, and manipulate large amounts of data concurrently, and in a consistent manner (plus allow for failure recovery, etc.).

Data was stored in databases, and the user had to perform operations explicitly to retrieve data from the system. The burden of retrieving relevant information was on the user. This is the traditional 'pull' paradigm, where the user retrieves information by performing an explicit action in the form of a query, application, or transaction execution.

Figure 1 indicates a different approach to information retrieval and management. In this push paradigm, the user does not have to query or retrieve information as it changes. The system is responsible for accepting user needs (in the form of situations

to monitor, business rules, constraints, profiles, continuous search queries, etc.) and informs the user (or a set of users) when something of interest happens. This paradigm relieves the user from frequently querying the data sources, and shifts the responsibility of monitoring from the user to the system. Of course, in order to accomplish this, the system needs to have additional functionality that is not part of traditional DBMSs. Although this mode of operation is recognized as beneficial and results in significantly less data transfers, accomplishing this for various architectures (such as distributed, federated, and network-centric) requires enhancements to the underlying system or needs to incorporate agents or mediators that can carry this out in a nonintrusive manner. In other words, the system needs to have the capability to selectively push information. This is a paradigm shift from the traditional information systems and also from the users' viewpoint.

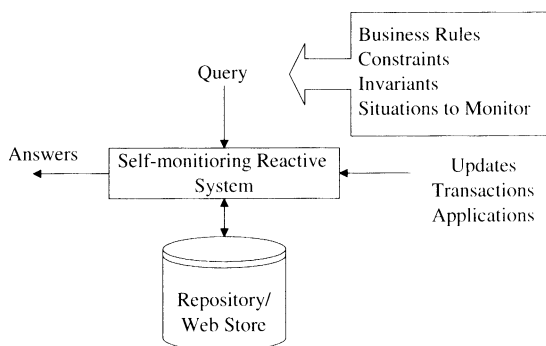


Fig. 1. Information retrieval using the push paradigm

### 3.1 Push-Based Architectures

Push technology can be introduced into a system in a number of ways. The approach primarily depends on the characteristics of the underlying system in terms of its openness. The following options can be inferred based on the underlying system characteristics.

**Integrated.** In this approach the underlying system is actually modified to incorporate the push technology in the form of ECA rules. This approach assumes that the source code for the underlying software is available and that the developers have sufficient understanding of the system to make changes at the kernel level. For example, the Sentinel object-oriented active system [3, 7] used this approach on the OpenOODB system from Texas Instruments. The sentry mechanism of the underlying system was extended to introduce notifications inside the wrapper for each method to detect primitive events. Once primitive events were detected, the composite events were detected and rules were executed outside of the underlying system. The primary advantage of

the integrated approach is its flexibility to add a minimum amount of code and incorporate many kinds of optimization that results in good performance. The footprint for primitive event detection is small. Some of the functionality needed for selective push technology (such as deferred action execution) can be easily incorporated using the integrated approach. So far, a number of research prototypes of active database systems have been developed, such as HiPAC [4], Ariel [16], Sentinel [2, 3], Starburst [23], Exact [12], Postgres [21], PEARD [1], SAMOS [14, 15] etc. Most of them are developed from scratch or integrated directly into the kernel of the DBMS. The integrated approach provides the following advantages [3]: it does not require any changes to existing applications, the DBMS is responsible for optimizing ECA rules, DBMS functionality is extended, and better modularity of applications is achieved and maintenance is easier.

However, the implementation of an integrated approach requires access to the internal workings of a DBMS into which the active capability is being integrated. This requirement of access to source code makes the cost of an integrated approach very high and requires a long integration time as well. Hence, most integrated systems are research prototypes.

**Agent-based/Mediated.** The assumption for this approach is that one does not have access to the source code of the underlying system. In fact, this is true in many real-life scenarios where a commercial-off-the-shelf (COTS) system is being used (relational DBMS is an example). However, the underlying system may provide some hooks, using which one can incorporate push capability effectively. We have experimented with this approach in a number of ways and have developed mediators/agents [17] to add full active capability to a relational DBMS. Intelligent agents are introduced between the end user (client) and the system (of course, transparently to the user), and the agent provides additional capabilities that are not provided by the underlying system.

**Wrapper-based.** For this approach, the assumption is that the underlying system is a legacy system and hence does not support appropriate hooks. Hence it is extremely difficult (and impossible in most cases) to modify the underlying source code. Typically, a wrapper is built that interfaces to the outside world, and push capabilities are added to this wrapper. The wrapper in turn uses the API of the underlying system and may add some additional functionality, not provided by the underlying system (sorting, for example). This approach needs a good understanding of the underlying system, and the wrapper has to be developed for each legacy system separately. This approach is not preferred unless it is the only alternative to bring the system on par with other systems and/or to bring the legacy system into a federation or a distributed environment.

## 4 WebVigIL Architecture

WebVigIL is a change detection and notification system that can monitor and detect changes to unstructured and semistructured documents in general. It provides a powerful way to disseminate information efficiently without sending unnecessary or

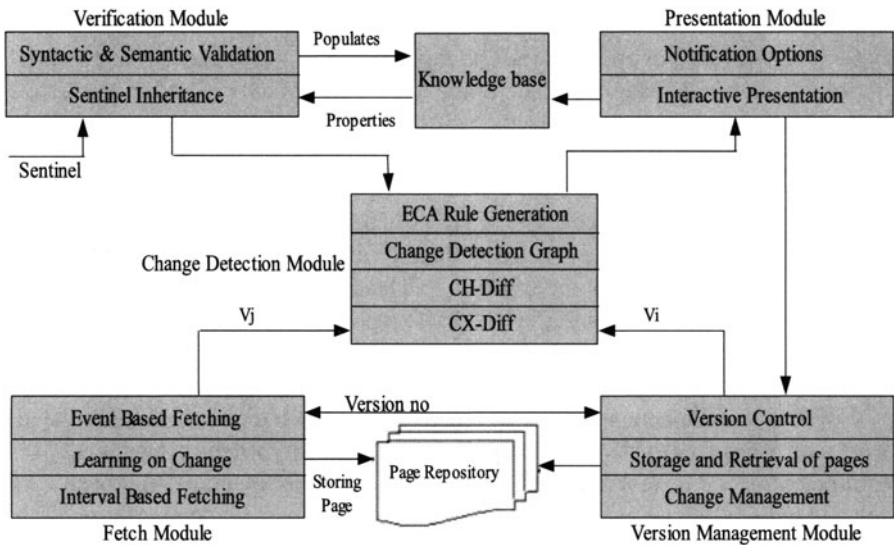


Fig. 2. WebVigIL architecture

irrelevant information. The emphasis in WebVigIL is to detect changes and notify the users based on user-defined profiles. The current work addresses HTML/XML documents that are part of the Web repository. WebVigIL aims at investigating the specification, management, and propagation of changes as requested by the user in a timely manner.

Figure 2 summarizes the high-level architecture of WebVigIL. Users specify their interest in the form of a *sentinel* that is used for change detection and presentation. Information from the sentinel is extracted, validated, and stored in a data/knowledge base and is used by the other modules in the system. The functionality of each module in the architecture is described briefly in the following sections.

4.1 User Specification

There is a need to define an expressive specification language using which the user can specify his/her policy for monitoring and notification of changes to Web pages. WebVigIL provides an expressive language with well-defined semantics for specifying the monitoring requirements of a user, pertaining to the Web. Each monitoring request is termed a *sentinel*. The syntax of the specification language is shown in Fig. 3. The semantics of the change specification language are briefly summarized below.

The language allows the specification of a sentinel in terms of previously defined sentinels. This provides a mechanism for tracking correlated changes. Hence, the sentinel target can be a URL of a Web page or a previously defined sentinel. In addition, the start and end of a sentinel may be based on other sentinels. The user can specify his/her change type of interest from a suite of change types at appropriate levels of granularity. Changes only at page level may be overkill in many cases. Hence,

WebVigIL supports customized changes like keywords, phrases, etc., pertaining to portions of a page. In addition, WebVigIL also supports detection of more than one change (composite changes) on a given page using binary operators AND and OR. Change detection using unary operator NOT is also supported. The specification

<Sentinel>	::=	Create Sentinel <sentinel-name> Using < sentinel-target> [Monitor < sentinel-type>] [Fetch <time interval>  on change] [From <time point>   <from event>] [To <time point>  <to event>] [Notify By <contact options>] [Every <time interval>   interactive  best effort   immediate] [Compare <compare options>]
<sentinel-name>	::=	Identifier
<sentinel-type>	::=	[<unary op>]<change type> [<binary op> <change type>]
<change type>	::=	any change  all links  all images   all words { except {<word1>,...<wordn>}}   table : {<table id> }   list : {<list id>}   phrase : {<phrase1>[, <phrase2>, ...<phrasen>]}   regular expression : {<exp>}   keywords : {<word1> [, word2 ,...wordn]}
<sentinel-target>	::=	sentinel<sentinel name> <url> [<binary op> sentinel < sentinel name> <url>]
<time interval>	::=	<integer>{second   minute  hour  day  week }
<time point>	::=	<month>/<day>/<year>[+ <time interval>]   Now [+ <time interval>]
<unary op>	::=	NOT
<binary op>	::=	AND   OR
<from event>	::=	start(<sentinel name>)[+ <time interval>]   during(<sentinel name>)   end(<sentinel name>)[+ time interval]
<to event>	::=	start(<sentinel name>)[+<time interval>]   end(<sentinel name>)[+ time interval]
<contact options>	::=	email <email address>  fax <fax no>  PDA <details>
<compare options>	::=	pairwise   moving<n>   every<n>
<n>	::=	integer

**Fig. 3.** Change specification language syntax

language also supports activation and deactivation of a sentinel. WebVigIL does not currently support continuous queries, but only interval-based monitoring requests. Hence, the user has to specify the start and end time of a sentinel. Users can specify the duration as one of the following: Now, Absolute time, Relative time, and Event-based time. 'Now' keeps track of the current time, and hence the current time (time of sentinel specification) is used. 'Absolute time'  $T$  can be specified as a fixed point on the time line. 'Relative time' is defined as an offset from a time point (either absolute or event-based). We assume that events, such as the start and end of a sentinel, can be mapped to specific time points and can be used to trigger the start or end of a new sentinel. The start of a sentinel can also be dependent on the active state of another sentinel and is specified by the event 'during'. The event 'during s' defines that a sentinel should be started in the closed interval of s and the start should be mapped to 'Now'.

To detect changes between two given versions of the same page, the versions of the page need to be fetched. Hence, the fetch frequency of a page has a bearing on the monitoring of the page. The user can monitor a page based on the actual change frequency, or at a user-specified frequency. The specification of the actual

change frequency relieves the user of knowing when the page changes and requests the system to do its best effort to fetch the page on modification. In this case, the system uses the heuristics-based best effort algorithm (BEA) as explained in [5] to fetch the pages.

Notification and propagation of changes is an important aspect of change monitoring. The mechanism selected for notification is important, especially when multiple types of devices with varying capabilities are involved. The ‘contact options’ allow the users to select the appropriate mechanism for notification from the following set of options: e-mail, fax, or PDA. In addition, when to notify the detected changes is also very important. The user can specify a user-defined time interval or may want to be notified as soon as a change is detected. The user may specify a notification frequency of ‘immediate’ or ‘best effort’ for this purpose. ‘Immediate’ indicates notification immediately on change detection. ‘Best effort’ is defined as notify as soon as possible after change detection. Hence, best effort is equivalent to immediate but has lesser priority than immediate for notification. WebVigiL uses the active capability to notify the users of the changes detected for immediate and best effort options. ‘Interactive’ is a notification approach where the user interactively explores the detected changes as and when needed. A WebVigiL dashboard is provided to the user to view and query the changes generated by his sentinels. The change detection module in WebVigiL creates a change detection graph (CDG) based on the user properties for detecting changes and notifies the user when a change is detected. The CDG is explained in detail in Sect. 4.3.

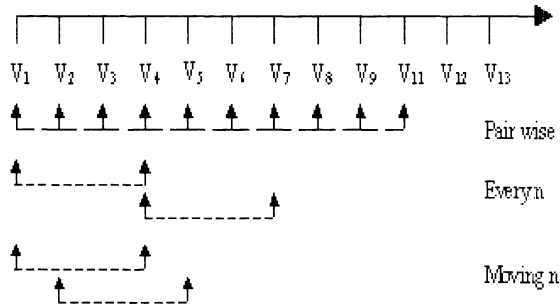


Fig. 4. Compare methods

To allow flexible monitoring of the pages, the user should have the privilege of choosing the page for reference. Given a sequence of versions  $V_1, V_2, \dots, V_n$ , of the same page, the user may be interested in knowing changes with respect to different references. Hence, the specification language provides multiple ways to compare changes. The various compare options are shown in Fig. 4. The default is ‘Pair wise’, which will allow change detection between two chronologically adjacent versions. ‘Every  $n$ ’ allows a user to detect changes between versions  $V_i$  and  $V_{i+n}$ . For the next comparison, the  $n$ th page becomes the reference page. ‘Moving  $n$ ’ is a moving window concept for tracking changes. For ‘Moving  $n$ ’, the first reference page is compared

with the  $n$ th page. For the next comparison, the subsequent version is taken as the reference page and is compared with  $n$ th page, starting from the reference page. For example, if a user specifies the compare option of 'Moving  $n$ ' where  $n=4$ ,  $V_1$  will be the reference page for  $V_4$ . The next comparison will be between  $V_2$  and  $V_5$ .

## 4.2 Knowledge Base

The details of a sentinel need to be stored (in a persistent and recoverable manner), as several modules use this information at run time. For example, the change detection module detects changes based on sentinel information such as the URL to be monitored, the change and compare specifications, and the start and end of a sentinel. The fetch module fetches the pages based on the user-specified fetch policy. The notification module requires appropriate contact information and a notification mechanism to notify the changes. User information, such as the sentinel installation date and the page versions for change detection and storage path of detected changes, also need to be stored to allow a user to keep track of his/her sentinels.

To satisfy all the above requirements, the metadata (the WebVigiL Knowledge Base (KB)) generated and used by different modules is stored in a relational DBMS. The monitoring request is parsed and sentinel properties are extracted, validated, and stored in the knowledge base.

## 4.3 ECA Rule Generation

Once the sentinel is registered the following operations are performed by the system: enabling/disabling of a sentinel (based on its start/end time), fetching the requested page, detecting changes of interest, and notifying the user(s) of the change. For every sentinel, the ECA rule generation module generates ECA rules [6, 7] to perform some of these operations. The ECA paradigm has been used for monitoring the database state in active databases and as a stand-alone system for monitoring objects in applications (both centralized and distributed [22]). As part of the Active Object-oriented system [2, 3], a local event detector (LED) [10] has been developed as a library that can be used to declare events and associate rules to be executed when events occur. Primitive events (as method executions) and temporal events (both absolute and relative time), as well as composite events (and, or, seq, and periodic used in WebVigiL) are supported in LED. ECA rules provide an elegant mechanism for supporting asynchronous executions based on events (temporal or otherwise).

**Activation/deactivation.** By default a sentinel is active during its lifespan but can be disabled explicitly by the user. The start/end of a sentinel can be time points or events (as explained in Sect. 4.1). When a sentinel's start time is 'now', it is enabled immediately. But in cases where the start is at a later time point or depends on another event that has not occurred, enabling of the sentinel is deferred until the time is reached or the event of interest is reached. To enable/disable sentinels appropriate events and rules are generated using LED.

Consider the scenario where sentinel *s1* is defined in the interval [12/02/02, 01/02/03]. At time 12/02/02, *s1* has to be enabled. The events and rules that are generated to enable *s1* at compile time are shown in Fig. 5. When the temporal event

Event Temp1	= createTemporalEvent(12/02/02)
Rule T1	= createRule(Temp1)
Event Start_s1	= createEvent("start_s1")
Event Fetch_s1	= createPeriodicEvent(Start_s1, 2, End_s1)

**Fig. 5.** Events and rules for *s1*

Event Start_s2	= createEvent("start_s2")
Rule R_start_s2	= createRule(Start_s1)

**Fig. 6.** Events and rules for *s2*

Temp1 is triggered, the associated rule T1 is executed, which in turn raises the event Start\_s1. Triggering of the event Start\_s1 activates the sentinel *s1*. If sentinel *s2* defined over the interval [start(*s1*), end(*s1*)] arrives, the following events and rules are generated as shown in Fig. 6. The rule R\_start\_s2 is associated with the event Start\_s1. This rule raises the Start\_s2 event when *s1* starts. In this manner, ECA rules are used to asynchronously activate and deactivate sentinels at run time. Once the appropriate events and rules are created, the local event detector handles the execution at run time. By enabling/disabling of a sentinel we mean addition/deletion of that sentinel in the change detection graph that is detailed in the next section.

**Change detection graph:** When a page is fetched, for every sentinel that is interested, changes are computed and notified. When more than one sentinel is interested in the same type of change on the same page, redundant change computation is avoided by grouping the sentinels on their change type and page. In order to represent this relationship, we construct a change detection graph. Consider the scenario where sentinel *S3* is specified on the Web page `www.cnn.com` for monitoring the change type 'all images'. Another sentinel *S1* is also specified for the same Web page but detects composite changes on 'all images' and 'all links'. The change detection graph constructed for the sentinels *S1* and *S3* is shown in Fig. 7. The different types of nodes in the graph are as follows:

- **URL node:** A URL node is a leaf node that denotes the page of interest.
- **Change type node:** All level-1 nodes in the graph belong to this category. This node represents the type of change on a page (all words, links, images, keywords, phrases, table, list, regular expression, any change).
- **Composite node:** A composite node represents a combination of change types. All nodes that are above level-1 in the graph belong to this type.

In the graph, to facilitate the detection and propagation of changes, the relationship between nodes is captured using the subscription/notification mechanism. The higher-level nodes subscribe to the lower-level nodes in the graph. This subscription information is maintained in the subscriber list at each node. The URL node contains the list of references to the change type nodes. All the higher-level nodes contain the



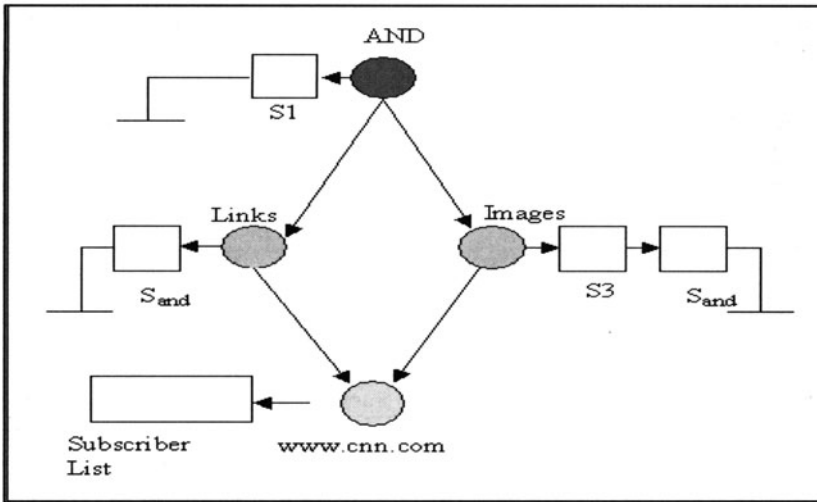


Fig. 7. Change Detection Graph

list of interested sentinels. Each sentinel has a subscriber that contains the references to the composite nodes.

At run time whenever a request for monitoring a page arrives, the corresponding nodes are created (if they do not already exist), and the sentinel is added to the change detection graph. When a page is fetched, the associated URL node is notified about the page. The URL node propagates this page to all the change type nodes that have subscribed to it. Finally, at the change-type nodes, the change is computed between the current page received and an appropriate reference page (based on the compare option) that is fetched from the page repository. If there is any change then the sentinels subscribed to it are notified. When this change type is part of a composite change, those composite nodes are also notified.

Consider the scenario where a page  $p_1$  changes twice a day and sentinel  $S_i$  is interested in both the changes taking place each day, whereas sentinel  $S_j$  is interested in changes once in two days. In this case, both the sentinels are interested in the same type of change on the same page but with different requirements (versions of page). For this, additional information is maintained at the change-type node in order to differentiate the change detection and change propagation among these sentinels.

As shown in Fig. 7, the arrows in the graph represent the parent-child relationship between the nodes. Sentinel  $S_1$  should be notified only when there is a change to links and images. Since change is computed only at level-1 nodes,  $S_1$  should have a representation of itself at the constituent change-type node. In order to facilitate this, a proxy sentinel  $S_{and}$  with the same properties as  $S_1$  is created on the links and images node. The composite node AND notifies  $S_1$  only when it receives notifications from both of its constituent sentinels.

**Change detection.** The change detection tool should be capable of detecting preferred change, such as the appearance/disappearance of objects of users' interest

on a page. Consider the following scenario: A student wants to monitor the college schedule of classes for a particular course name (keyword). In such cases, detecting changes to the complete page results in excessive computation and dissemination of irrelevant information. Hence, there is a need to support detecting changes based on user’s intent. We view a page as a sequence of words and certain content-defining tags while ignoring other presentation tags. Users may be interested in the appearance or disappearance of certain words or a section of contiguous words or links/images in the page. Thus the contents of a page can be classified, based on the user intent, into

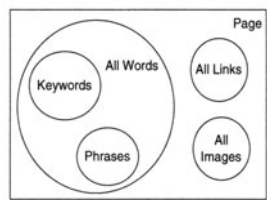


Fig. 8. Classification of page

Change Type	Synopsis	Approach
Links	Insertion of new links and deletion of old links	CH-Diff
Images	Insertion of new images or deletion of old images	CH-Diff
Keyword(s)	Insertion or deletion of selected words	CH-Diff
Phrase(s)	Insertion/deletion/update to selected text phrase	CH-Diff
All Words	Any change to words in the page	LCS

Fig. 9. Synopsis

keywords, phrases, all-words, links and images.

- Keywords: a set of unique words from the page
- Phrase: contiguous words from the page (no upper bound)
- All-Words: all the words in the page constitute this set
- Links: a set of hypertext references extracted from page
- Images: a set of image references extracted from the image source

The classification of the page based on user intent is shown in Fig. 8. Appearance or disappearance of these content-based objects of interest can flag a change. The synopsis of the various changes detected by WebVigiL is shown in Fig. 9. As the above-defined changes are content-based, we do not detect changes to the structure of the document. The structure of the document is only considered for efficient change detection to the contents. WebVigiL supports change detection to HTML and XML, which are the standard formats for electronic publishing on the Web. Over the coming years XML is likely to replace HTML as the standard Web publishing language, but until then both will coexist. In an XML page, a combination of the content and the tags define the nature of the content, whereas in HTML they define the presentation aspects of the content. Hence, as the format and representation of both HTML and XML differ, separate approaches need to be adopted for change detection for these documents. These change detection techniques are discussed in detail in [20].

4.4 Event-Based Fetching

For monitoring a page, after every time interval *t*, we check for any change in page properties, such as the last modified time stamp of the page, and then actually fetch the

page if there is a change. Hence we need a mechanism for triggering the monitoring requests at a given time point after the lapse of interval  $t$ . We solve this problem by using the notion of a periodic event [6]. A periodic event is an event that repeats itself with a constant and finite amount of time. The specification for a periodic event is  $\text{PeriodicEvent}(E1, [t], E3)$  where  $E1$  and  $E3$  are the events (or time specifications) that act as initiator and terminator, and  $t$  is the time interval. In WebVigiL,  $E1$  and  $E3$  are the start and end events of a sentinel and  $t$  is the interval at which the page should be monitored. To actually fetch the page, we associate a fetch rule with this periodic event. This fetch rule performs the functionality of the monitoring request, i.e., it fetches the page based on changes to the page properties.

Consider the scenario where sentinel  $s1$  specifies the fetch time as 2 days. The periodic event generated for  $s1$  is as follows:  $\text{Event Fetch\_s1} = \text{createPeriodicEvent}(\text{Start\_s1}, 2, \text{End\_s1})$ , where  $\text{Start\_s1}$  and  $\text{End\_s1}$  are the start and end events of sentinel  $s1$ . The event  $\text{Start\_s1}$  initiates the periodic event. For every interval  $t$  (2 days) the periodic event is raised until event  $\text{End\_s1}$  occurs. For sentinels that explicitly specify the polling interval, we generate a unique periodic event. The interesting and difficult case is when the user expects the system to notify him/her as and when the page changes. In such cases, the system is required to tune its polling interval with the change frequency of the page. For this, we generate a common periodic event for all the sentinels interested in the same page. This rule achieves the required tuning by changing the interval of the periodic event base on learning.

When the rule is fired, it checks (condition part of the rule) for a change in metadata of the page and fetches the page (action part of the rule) if there is a change. Thus the periodic event controls both the polling interval and the lifespan of the fetch process. By metadata of the page, we mean page properties such as the page size, last modified time stamp, and checksum of the page. A fetch cycle for a page is triggered only when there is a change between the metadata of the current version of the page and that of the previous version. Depending upon the nature (static/dynamic) of the page being monitored, the complete set or subset of the metadata is used to evaluate the change. The different types of request used for obtaining the metadata of the page are:

- For static pages, the HTTP HEAD request is used to obtain the metadata of the page. A change in time stamp of the page with an increase or decrease in page size is flagged as change, and the page is fetched.
- If time stamp is modified, but the page size is unchanged, the HTTP GET request is used to retrieve the page, and the checksum of the page is calculated. The page is added to the page repository only if the calculated checksum differs from the checksum of the previously cached copy of that page.
- For pages that are not provided with last modified timestamp, such as dynamically generated pages or cases where previous attempts to retrieve page properties have failed, the HTTP GET request is used to retrieve the page. A change is then flagged by calculating the checksum.

**Types of fetch rules.** A fetch rule is created and used for polling the page of interest specified in the sentinel. Similarly to the change specification, a user can specify a sentinel for fetching with two options: ‘On Change’ or ‘Interval Based’. Based on the

option specified in the sentinel, The event-based fetch module generates a best-effort (BE) rule or an interval-based (IB) rule. These rules differ in the way they handle the  $t$  (fetch interval) of the periodic event.

- **Best-effort rule:** In situations where the user has no information about the change frequency of a page, it is necessary to tune the fetch frequency to the actual change frequency of a page. BE rules use a best-effort algorithm (BEA) [5] to achieve this tuning. In the BEA, the next fetch interval ( $P_{next}$ ) is determined from the history of changes to that page. When the next polling interval is determined, the BE rule changes the interval  $t$  of the periodic event. Clearly, the effectiveness of the algorithm depends on the accurate estimation of the fetch interval. The event-based fetch module generates a BE rule  $BE_i$  for every unique page  $u_i$ , and maps other sentinels with fetch option ‘On Change’ on  $u_i$  to the generated rule  $BE_i$ .
- **Interval-Based rule:** The user can explicitly specify a fetch frequency. For example, Don may know that a page is changing every 4 hours starting at 9.00 am, and hence can specify a sentinel to start monitoring the page with a fetch interval of 4 hours. For this, a periodic event whose periodicity (interval  $t$ ) equals to the given interval is created, and an IB rule  $IB_i$  is associated with it to fetch the page. As a result there will be more than one IB rule on a given page with different or same periodicity, where each rule is associated with a unique periodic event (i.e., with different start and end times).

A BE rule and many IB rules can be set for the same page. Thus, there can be situations where both a BE rule and an IB rule fetch the same version of the page resulting in multiple copies of the same version in the page repository. To avoid this situation, we synchronize the fetching with respect to the last fetched version of the page. A rule initiates the fetch process only when there is no version  $V_i$  of the page  $u$  with last-modified-timestamp (LMT) equal to the LMT of the page it is required to fetch.

#### 4.5 Caching and Management of Pages

A server-based repository service archives and manages versions of pages monitored by the sentinels. The primary purpose of the repository service is to reduce the number of network connections to the remote Web server, thereby reducing network traffic. When a remote page fetch is initiated, the repository service checks for the existence of the remote page in its cache, and if present, the latest version of the page in the cache is returned. In cases of cache miss, the repository service requests that the page be fetched from the appropriate remote server.

Since all the versions of a particular page have to be stored in the cache, each URL has to be mapped to a unique directory. The complete URL cannot be used as a directory name since the length of the URL is large in many cases. Below we discuss two approaches to establish the required mapping for the directory structure. We have analyzed the approaches experimentally to determine which one scales up for tens of thousands of pages. The approaches are discussed below.

**Hash-based approach.** Each unique URL is inserted into a hash table with URL as the key. This unique value represents the directory where the corresponding versions are stored. Consider two URLs  $x/y/z/i.htm$  and  $x/y/z/j.htm$ , which have common path information  $x/y/z$ . Instead of hashing the whole URL twice, we generate a mapping for the common path once and reuse the same mapping whenever required. This is achieved by maintaining two separate mappings for the path and file name. For example, the URLs `ranger.uta.edu/~cook/aa/lectures/l10/L10.html` and `ranger.uta.edu/~cook/aa/lectures/l10/node14.html` will be mapped to

D1F1 and D1F2 where D1 is the value mapped to `ranger.uta.edu/~cook/aa/lectures/l10/`, and F1, F2 mapped to files `L10.htm` and `node14.htm`.

**Directory-based approach.** In this approach the path of the URL is replicated for the directory structure. For example, given the URL `ranger.uta.edu/~cook/aa/lectures/l10/L10.html`, the directory structure is maintained up to the file name. In some cases (dynamically generated pages) the length of the file name can be very large. As the operating system imposes a restriction on the length of the directory name, a hash-based approach is adopted to generate a unique mapping for the file name. For the previous example `L10.html` is mapped to F1.

**Experimental evaluation.** We evaluate the two approaches based on the time taken to construct the mapping and the time taken to reconstruct the mapping (in case of recovery) plus the time to retrieve the page given its URL. For the hash-based approach we use the java hash provided in Java 1.4. The selection of data sets for the experiments was based on the length of the path (excluding the file name) of the URL, denoted as the depth and the number of URLs. The data sets are represented as L#, where # denotes the number of URLs with depth varying between 1 and 3, and M# with depth varying between 4 and 6. Figure 10 shows the time taken to

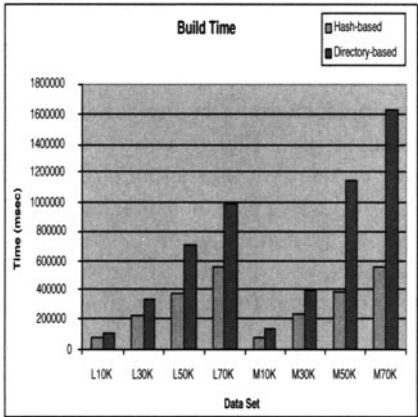


Fig. 10. Build time

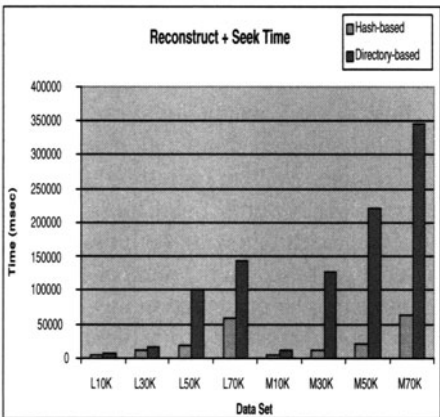


Fig. 11. Reconstruction + seek time

build the mapping for each of the data sets. The directory-based approach takes more time than hash-based approach as more I/O is involved. Figure 11 shows the time taken to rebuild the mapping plus the time taken to retrieve the page given the URL. Since the mapping is made persistent as and when unique URLs are hashed (during the build time), during reconstruction the unique mappings for persistent URLs are not regenerated, thus saving time. It can be observed that even though there is no reconstruction required in the directory-based approach except for the file mapping, the time taken is much larger. This increase in time is because the time taken to retrieve the pages increases with increase in depth, as more directories have to be traversed, as is apparent from Fig. 11.

#### 4.6 Change Presentation

Change presentation is the last phase of Web monitoring, where the changes detected, as outlined in the previous sections, are presented to the user. For meaningful interpretation of the presented changes, we investigated three ways to present it to the user:

- *Only-change approach*: Showing only the changes and omitting the common objects of the two pages is advantageous for pages of large size but will make interpretation intricate. This approach can be meaningful for hand-held devices to conserve the amount of data transmitted over a limited bandwidth.
- *Single-frame approach*: Produce a single document by merging the two documents summarizing all inserted, deleted, and common objects. The advantage lies in displaying the common objects just once, but with the drawback of possibly changing the page structure.
- *Dual-frame approach*: Showing both the documents side-by-side in different frames and highlighting the changes between the documents has the advantage of easy interpretation of the changes presented. When the number of changes to be presented is large, this approach may make it difficult to interpret the changes. This can be remedied by presenting parts of the pages at a time to limit the number of changes displayed in each installment.

In WebVigil we intend to use all of the three presentation schemes summarized above in a selected combination depending upon the type of change being presented. For example, we plan on testing the dual-frame approach for presenting changes to phrases and keywords. For displaying changes to images, we plan on using the single-frame approach (showing both the old and new image). Finally, for the change type any-change, based on the number of changes detected, we use a heuristic cost model for choosing the presentation mechanism between the dual-frame, only-change and single-frame approach for displaying changes.

## 5 Conclusions and Future Work

WebVigil is a change monitoring system for the Web that supports specification and management of sentinels and provides presentation of detected changes in multiple

ways (batch, interactive, for multiple devices). The first prototype has been completed and includes the following features: Web-based sentinel specification [5], ECA rule-based fetch that includes learning [5] to reduce the number of times a page is fetched, population of the knowledge base, and detection of changes to HTML and XML pages [20]. A simple presentation module for the schemes briefly outlined in this paper has been implemented.

Currently, the individual modules are being integrated to instrument the first version of a complete WebVigIL system. The performance evaluation of change detection algorithms and their comparison with other approaches are also currently underway.

WebVigIL has brought out a number of problems that are not yet completely addressed. The implementation of WebVigIL has raised a number of system-related issues, such as where to use active capability effectively, persistence of large numbers of pages, scalability issues, and dealing with a number of devices for presentation and notification. A number of synchronization issues between the modules arose during development of the prototype. On the conceptual side, a number of open problems remain, such as change detection of dynamic pages, position-independent change detection, adaptive algorithms to minimize fetch, and change detection on multiple pages. Finally, currently a sentinel has a single lifespan and future work will include supporting multiple life spans for a sentinel.

## 6 Acknowledgements

The authors would like to thank Alpa Sachde, Shravan Chamakura, and Ajay Eppili for their help in producing this chapter, proofreading the contents, and adding the author and subject indexes. This work was supported in part by the Office of Naval Research, the SPAWAR System Center San Diego and by the Rome Laboratory (grant F30602-02-2-0134), and by NSF (grants IIS-0123730 and ITR-0121297).

## References

1. J. Alexander, S. D. Urban, and S. W. Dietrich. PEARD: A prototype environment for active rule debugging. *Intelligent Information Systems: Integrating Artificial Intelligence and Database Technologies*, 7(2):111–128, October 1996.
2. E. Anwar, L. Maugis, and S. Chakravarthy. A new perspective on rule support for object-oriented databases. In *Proceedings, International Conference on Management of Data*, pages 99–108, Washington, DC, May 1993.
3. S. Chakravarthy, E. Anwar, L. Maugis, and D. Mishra. Design of Sentinel: An object-oriented DBMS with event-based rules. *Information and Software Technology*, 36(9):559–568, 1994.
4. S. Chakravarthy et al. HiPAC: A research project in active, time-constrained database management, final report. Technical Report XAIT-89-02, Xerox Advanced Information Technology, Cambridge, MA, Aug 1989.
5. S. Chakravarthy et al. WebVigIL: Architecture and functionality of a Web monitoring system. Technical Report CSE-2003-5, CSE Department, University of Texas, Arlington, TX 76019, 2003.

6. S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, Contexts, and Detection. In *Proceedings, International Conference on Very Large Data Bases*, pages 606–617, 1994.
7. S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data and Knowledge Engineering*, 14(10):1–26, October 1994.
8. Changedetection. <http://changedetection.com/>.
9. G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proceedings, International Conference on Data Engineering*, San Jose, CA, 2002.
10. R. Dasari. Design and implementation of a local event detector in Java. Master's thesis, Database Systems R&D Center, CIS Department, University of Florida, E470-CSE, Gainesville, FL 32611, 1999.
11. P. Deolasee et al. Adaptive push–pull: disseminating dynamic Web data. In *Proceedings of the Tenth International WWW Conference*, Hong Kong, China, 2001.
12. O. Diaz, N. Paton, and P. Gray. Rule management in object-oriented databases: A unified approach. In *Proceedings 17th International Conference on Very Large Data Bases*, Barcelona (Catalonia, Spain), Sept 1991.
13. F. Dougliš, T. Ball, Y. F. Chen, and E. Koutsofios. The AT&T internet difference engine: Tracking and viewing changes on the Web. *World Wide Web Journal*, 1(1):27–44, January 1998.
14. S. Gatzia and K. R. Dittrich. SAMOS: an active, object-oriented database system. *IEEE Quarterly Bulletin on Data Engineering*, 15(1-4):23–26, December 1992.
15. S. Gatzia and S. R. Dittrich. Rules in database systems. In N. Paton and M. Williams, editors, *Events in a Active Object-Oriented System*, pages 127–142. Springer, Berlin Heidelberg New York, 1993.
16. E. N. Hanson. The Ariel project. In J. Widom and S. Ceri, editors, *Active Database Systems — Triggers and Rules for Advanced Database Processing*, pages 64–86. Springer, Berlin Heidelberg New York, Berlin, 1996.
17. L. Li and S. Chakravarthy. An agent-based approach to extending the native active capability of relational database systems. In *Proceedings, International Conference on Data Engineering*, Sydney, Australia, 1999.
18. L. Liu, C. Pu, and W. Tang. WebCQ: Detecting and delivering information changes on the Web. In *Proceedings of International Conference on Information and Knowledge Management (CIKM)*, Washington DC, 2000.
19. B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda. Monitoring XML data on the Web. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2001.
20. N. Pandrangi et al. WebVigiL: User-profile based change detection for HTML/XML documents. In *Proceedings 20th British National Conference on Data Bases*, Coventry, UK, 2003.
21. M. Stonebraker and G. Kemnitz. The Postgres next-generation database management system. *Communications of the ACM* 34, 34(10):78–92, 1991.
22. W. Tanpisut. Design and implementation of event based subscription–notification paradigm for distributed environments. Master's thesis, CSE Department, University of Texas, Arlington, TX, 2001.
23. J. Widom. The Starbust Rule System. In J. Widom and S. Ceri, editors, *Active Database Systems — Triggers and Rules for Advanced Database Processing*, pages 87–110. Springer, Berlin Heidelberg New York, 1996.
24. Xyleme. <http://xyleme.com/>.



---

# DREAM: Distributed Reliable Event-Based Application Management

Alejandro Buchmann, Christof Bornhövd\*, Mariano Cilia\*\*, Ludger Fiege, Felix Gärtner\*\*\*, Christoph Liebig†, Matthias Meixner, and Gero Mühl‡

Databases and Distributed Systems Group  
Department of Computer Science  
Darmstadt University of Technology, Germany  
buchmann@informatik.tu-darmstadt.de

**Summary.** New applications and the convergence of technologies, ranging from sensor networks to ubiquitous computing and from autonomic systems to event-driven supply chain management, require new middleware platforms that support proactive event notification. We present a system overview and discuss the principles of DREAM, a reactive middleware platform that integrates event detection and composition mechanisms in a highly distributed environment; fault-tolerant and scalable event notification that exploits a variety of filter placement strategies; content-based notification to formulate powerful filters and concept-based notification to extend content-based filtering to heterogeneous environments; middleware-mediated transactions that integrate notifications and transactions; and scopes, which are administration primitives for both deployment- and runtime configurability, as well as for the management of policies. We discuss four prototypes that were implemented as proof-of-concept systems and present lessons learned from them.

## 1 Introduction

We are experiencing a convergence of technologies that results in an explosion of information and requires new paradigms for data and information management and processing.

- The World Wide Web is a huge source of information that was conceived for interactive search by humans. To exploit the Web in a mode other than human browsing, we are confronted by a need for filtering and interpreting a large amount of heterogeneous and often short-lived data.

---

\* IBM Almaden Research Center, USA

\*\* also Faculty of Sciences, UNICEN, Argentina

\*\*\* EPFL, Switzerland

† SAP AG, Germany

‡ TU Berlin, Germany

- The deployment of smart devices requires the continuous monitoring of events as well as the appropriate context information to interpret them properly.
- The miniaturization of sensors and their ubiquitous deployment will result in massive amounts of sensor signals that must be processed, often in real time.
- Huge distributed systems must be capable of detecting and correcting failures and return autonomously to stable operation.
- New business strategies, such as *event-driven supply chain management* and *zero-latency enterprise*, depend on the timely dissemination of information and business events.

Common to the above is that signals and data, which we abstract into the notion of *events* and *event notifications*, will flow to us. When dealing with streams of events our traditional, pull-based access mechanisms to stagnant data no longer work. Compare traditional pull-based data processing, where queries are issued against a database, with drinking water with a straw from a glass. The same straw (and the pull mechanism) is useless when trying to drink from the garden hose.

To understand the fundamental difference between traditional applications and the scenarios described above we will analyze the information space and its interactions. There are two major dimensions that characterize an interaction pattern (see Table 1): who initiates an interaction and the knowledge that exists about the counterpart. Along the first dimension we distinguish the requestor of a service or consumer of a unit of information from the provider of a service or information. Along the second dimension we distinguish between having knowledge of the identity of one's counterpart and having no knowledge.

In the well-known request–reply interaction pattern, the interaction is initiated by the client or consumer of information, and the request is directed at a specific server or information provider. If the identity of the service provider is unknown we have a case of anonymous request–reply. On the other hand, if the interaction is initiated by the producer of information, we have the cases depicted in the right column of the table. If the producer knows the identity of its counterpart, we have a classical messaging interaction. However, if the producer does not know the consumer a priori, we have the typical event-based interaction that depends on a mediator or broker to connect the interested parties.

		Initiator of interaction	
		Consumer	Producer
Knowledge of counterpart	Yes	<i>Request/Reply</i>	<i>Messaging</i>
	No	<i>Anonymous request/Reply</i>	<i>Event-based</i>

**Table 1.** Taxonomy of information processing space

While traditional applications in which the user is in control or where the responsiveness of the system is not critical are well served by user-initiated request-reply interactions, automated processes in distributed environments must react to (possibly large quantities of) events and cannot rely on consumer-initiated processing. In event-based systems the producers of the events can be unaware of the consumers, thereby allowing a loose coupling between producers and consumers. This facilitates the evolution of the system, since new applications that react to events can be added without affecting the deployed infrastructure. The inherent reconfigurability of event-based systems directly addresses volatile environments, which require agile applications and infrastructures, as was noted, for example, for enterprises that want to expedite their crucial business processes [90].

The fundamental difference in the interaction pattern found in emerging applications suggests that slowing down the flow of events to use traditional tools, e.g. through the use of caches or databases, is only a patchwork solution. It is our contention that a reliable infrastructure for management of streaming information is needed and that the importance of this infrastructure will increase as we move to a world populated by huge amounts of interconnected devices, services, and applications with different capabilities that will react and automate processes on our behalf. Streaming events must be detected, interpreted, aggregated, filtered, analyzed, reacted to, and eventually disposed of.

In this chapter we present the architecture and an overview of the components of DREAM, a distributed reactive middleware layer that consists of a publish/subscribe event notification service and the reactive capabilities to consume and react to events in heterogeneous environments. Specific research results and the details of the individual components of DREAM have been published elsewhere, and the reader is referred to these publications throughout the text. Here we concentrate on the motivation, the principles, the design decisions and their consequences, and we will illustrate the feasibility of our approach with four implemented prototypes.

We want to emphasize that different applications may have different notions of events and will stress different portions of the middleware. Therefore, we do not propose a one-size-fits-all platform. However, the interaction principles we discuss here are common to many application domains. We present here the issues and one possible incarnation of a publish/subscribe reactive middleware.

## 2 Related Work

A reactive middleware platform draws on many existing technologies. Accordingly, the related work is vast, and therefore is organized according to the technologies involved. We discuss in Sect. 2.1 data and event dissemination. Sections 2.2 and 2.3 deal with event and data aggregation and integration. Section 2.4 is devoted to the reactive functionality needed to respond to events. Section 2.5 discusses work related to fault tolerance and the integration of notifications and transactions. Finally, Sect. 2.6 addresses software engineering and management issues.

## 2.1 Event/Data Dissemination

Early work on broadcast disks addressed the issues of push-based information dissemination in asymmetric communication environments [2]. Under this approach data is simply broadcast (pushed) to all consumers. Notification services (also called event notification services) are in charge of propagating data/events to interested consumers. For instance, in CORBA an event service [74] was introduced to provide a mechanism for asynchronous interaction between CORBA objects. Here, an event channel acts as a mediator between suppliers and consumers of events. To overcome deficiencies of this service specification, the notification service [75] was proposed as a major extension with support for quality of service specifications and basic event filtering.

The Java Message Service (JMS) [56] provides the Java technology platform with the ability to process asynchronous messages. JMS was originally developed to provide a common Java interface (API) to legacy message-oriented middleware (MOM) products like IBM Websphere MQ (formerly known as IBM MQ-Series) or TIB/Rendezvous. In this way, the JMS API provides portability of Java code, allowing the underlying messaging service to be replaced without affecting existing code. JMS provides two models for messaging among clients: point-to-point (using a queue) and publish/subscribe (by means of topics). JMS has been part of Java Enterprise Edition (J2EE) since its origin, but it was incorporated as an integral part of the Enterprise Java Beans (EJB) component model in the EJB 2.0 specification. It includes a new bean type, known as message-driven bean (MDB), which acts as a message consumer providing asynchrony to EJB-based applications.

In recent years, academia and industry have concentrated on publish/subscribe mechanisms because they offer loosely coupled exchange of asynchronous notifications, facilitating extensibility and flexibility. The channel model has evolved to a more flexible subscription mechanism, known as subject-based, where a subject is attached to each notification [79]. Subject-based addressing features a set of rules that define a uniform name space for messages and their destinations. This approach is inflexible if changes to the subject organization are required, implying fixes in all participating applications.

To improve expressiveness of the subscription model the content-based approach was proposed, where predicates on the content of a notification can be used for subscriptions. This approach is more flexible but requires a more complex infrastructure [20]. Many projects in this category concentrate on scalability issues in wide-area networks and on efficient algorithms and techniques for matching and routing notifications to reduce network traffic [41, 72, 81]. Most of these approaches use simple Boolean expressions as subscription patterns and assume homogeneous name spaces.

More recently, a new generation of publish/subscribe systems that are built on top of an overlay network has emerged. This is the case of Scribe [89], which is restricted to topic-based addressing and is implemented on top of Pastry. The mapping of topics onto multicast groups is done by simply hashing the topic name. Hermes [85] uses a similar approach, also based on Pastry. Additionally, the system tries to get around the limitations of topic-based publish/subscribe by implementing a so-called

‘type- and attribute-based’ publish/subscribe model. It extends the expressiveness of subscriptions and aims to allow multiple inheritance in event types. A content-based addressing on top of a dynamic peer-to-peer network was proposed in [95] where the efficient routing of notifications takes advantage of the topology graph underneath. This work combines the high expressiveness of content-based subscriptions and the scalability and fault tolerance of a peer-to-peer system.

## 2.2 Event Aggregation

Events and associated data can be aggregated according to aggregation operations. In the context of active databases (aDBMSs) event aggregation involves the occurrence of two or more events. Complex situations can be specified in order to aggregate information from ‘low-level’ events into more complex ones. These are known as composite events and are usually expressed using an event algebra, such as those defined in HiPAC [35], Ode [50], SAMOS [48], or Snoop [24]. Such algebras require an order function between events to apply event operators (e.g. sequence), or to consume events. To determine which of these events should be consumed or selected, different consumption modes were defined [27]. Usually, events are point-based and time-stamped to provide a time-based order with the purpose of facilitating event selection. However, in open distributed environments global time is not applicable.

In [60], Lamport presented the happened before relation, which defines a partial ordering of events based on the causality principle. An event  $a$  happened before an event  $b$  (depicted  $a \rightarrow b$ ) if  $a$  could have influenced  $b$ ;  $a$  and  $b$  are said to be causally dependent. If neither  $a \rightarrow b$  nor  $b \rightarrow a$ , the events are said to be concurrent and causally independent. A system of logical clocks is introduced that assigns a natural number to each event (logical timestamp). Logical clocks are consistent with causality [91]: if  $a \rightarrow b$ , then  $a$ ’s timestamp is smaller than  $b$ ’s timestamp – the contrary is not true. In [91] the concept of vector time is presented, and it is shown that vector time characterizes causality: two events are ordered by vector time iff they are causally dependent. However, neither logical clocks nor vector clocks can deal with causal relations that are established through hidden channels and also cannot represent timed real-world events. Thus they are not appropriate for open systems.

An approximation for modeling the clock imprecision in distributed systems has been proposed. Assuming a sparse time base (where the points at which events can be generated are discretized and predetermined), Kopetz [58] proposed the *2g-precedence* model. This model establishes that if events are at least two time granules apart, the sequence of these events can be determined unequivocally. Here an upper bound to the precision is assumed and a virtual clock granularity  $g$  is defined. Since the granularity depends on the assumed precision, it is not a feasible approach for wide area networks and open distributed systems.

Schwidorski [92] adopted the 2g-precedence model to deal with distributed event ordering and composite event detection. She proposed a distributed event detector based on a global event tree and introduced 2g-precedence-based sequence and concurrency operators. However, event consumption is nondeterministic in the case of

concurrent or unrelated events. Additionally, the violation of the granularity condition (2g) may lead to the detection of spurious events.

Dyreson [38] proposed the notion of valid time indeterminacy for temporal databases, to model the fact that it is not known exactly when an event occurred. An interval timestamp together with a probability distribution is suggested to represent the time span during which the event is supposed to have occurred. As a consequence, querying the temporal database eventually results in multisets that represent alternative answers to the query.

In [63] an approach for timestamping events in large-scale, loosely coupled distributed systems is proposed. This uses accuracy intervals with reliable error bounds for timestamping events that reflect the inherent inaccuracy in time measurements.

Many projects on event composition in distributed environments such as [67, 51, 100] either do not consider the possibility of partial event ordering or are based on the 2g-precedence model. Therefore, they suffer from one or more of the following drawbacks [63]: they do not scale to open systems, they provide the possibility of spurious events, or they present ambiguous event consumption.

Systems that support composite events must also address the semantic issues associated with processing composite events. For example, how timestamps are generated and the way in which events are selected and consumed. Several recent projects have dealt with queries on streaming data [19, 5, 69, 26]. They are basically data flow systems where tuples flow through an acyclic directed graph of processing nodes that apply stream operators. These systems are mostly centralized and they monitor and aggregate data. The aggregated data serves as the triggering event.

### 2.3 Event Integration

As has been clearly identified, additional semantic metadata for the exchange of data or messages among independent applications or services is needed, not only in the context of B2B frameworks like ebXML [39], BizTalk [68], or RosettaNet [88] but also by the W3C in efforts like Semantic Web [6], DAML+OIL [33], or OWL [96]. In the first case, XML [14] and XML Schema [42] are used to define common vocabularies to describe data and business processes. Other data models similar to XML include OEM [83] and the models described in [1, 36].

In the context of W3C's Semantic Web initiative, RDF [61] and RDF Schema [15] are used to provide additional semantic metadata to better enable computer and users to exchange and integrate data. RDF provides an infrastructure that supports the representation and exchange of structured metadata to describe Web resources, like (parts of) Web pages, or other RDF metadata. RDF allows the description of properties of and interrelationships among those resources in terms of  $\langle \text{resource, attribute, value} \rangle$  triples. The attributes used can be declared in RDF Schemas which, similar to XML Schemas, give information about their intended meaning and specify restrictions on their values. RDF Schemas and XML Schemas can play a role similar to ontologies as a common semantic basis for data and metadata representation.

In our framework we use the MIX model [11, 12] for the representation of data (i.e. event content). Like XML/XML Schema or RDF/RDF Schema, MIX provides a

flexible representation model for data plus additional metadata based on a common domain-specific vocabulary. However, in addition to the functionality provided by the data models discussed above, MIX directly supports data integration by making the concept of semantic context (i.e. the explicit description of implicit assumptions about the meaning of the data) and conversion functions (which allow the automatic conversion of data/events from different sources to a common context) first-class citizens of the model itself. MIX should not be seen as an alternative to the models being developed in the context of the W3C but as a complementing approach that provides features that may find their way into the other XML-based models and standards.

## 2.4 Reactive Functionality

Reactive mechanisms were introduced in the late 1980s in the form of event-condition-action rules (ECA rules) in active databases [84]. The goal of active databases was to avoid unnecessary and resource-intensive polling in monitoring applications where events are detected as changes to a database and the application reacts to the occurrence of these events.

Reactive functionality is used to support a wide spectrum of applications ranging from workflow management [51, 62], personalization [66, 31], alerters [23], business rule enforcement [3, 98] up to the internal support for relational and object-oriented databases [99], particularly with the purpose of supporting integrity of constraints, view maintenance, access control, etc. Most recently, active functionality has been used in the context of XML repositories [10, 4].

Active database functionality developed for a particular DBMS became part of a large monolithic piece of software (the DBMS). Monolithic software is difficult to extend and adapt. Moreover, active functionality tightly coupled to a concrete database system hinders its adaptation to today's Internet applications, such as e-commerce, where heterogeneity and distribution play a significant role but are not directly supported by traditional (active) database systems [49].

Another weakness of tightly coupled aDBMSs is that active functionality cannot be used on its own without the full data management functionality. However, active functionality is also needed in applications that require no database functionality at all, or that require only simple persistence support. As a consequence, active functionality should be offered not only as part of the DBMS but also as a separate service that can be combined with other services to support, among others, Internet-scale applications.

Unbundling active databases consists of separating the active part from active DBMSs and breaking it up into components providing services like event detection, rule definition, rule management, and execution of ECA rules on the one hand, and persistence, transaction management, and query processing services on the other [49]. Afterwards, only necessary components can be rebundled in order to provide the required functionality. A separation of active and conventional database functionality would allow the use of active capabilities depending on given application needs with-

out the overhead of components that are not needed. Various projects like C<sup>2</sup>offein [59], FRAMBOISE [46], and NODS [32] have followed this approach.

From our point of view, unbundling active functionality from a concrete system and then rebundling the corresponding components in an open distributed environment is questionable. Unbundling in this context means to give up the ‘closed world’ assumption that traditionally underlies a DBMS. Inherent characteristics of open distributed environments impose new requirements that were not considered in centralized environments, such as the lack of global time, independent failures of nodes or communication channels, message delays, etc. The consideration of these characteristics has an enormous impact on the event detector [63], which is the essential component of an aDBMS [17]. In [16] cross-effects and potential incompatibilities arising from the combination of selective features of active, real-time, and distributed object systems are discussed.

## 2.5 Notifications and Transactions

Transactions are a well-known concept to provide reliability of execution and have been well studied in research [8, 52, 97]. In database systems, data access operations are grouped into logical units of work and are executed under transaction control. Database transactions guarantee atomicity, consistency, isolation, and durability (ACID) for the execution of operations in a unit of work, despite concurrency and despite the presence of application errors and system failures. Thereby, the programmer is shielded from a variety of complex situations that otherwise arise in the presence of concurrency and the possible multitude of failure modes. Finally, it is this clear separation of responsibilities into application-specific logic and generic middleware services that renders the concept of *transaction* so powerful [97].

In aDBMS, the concept of coupling modes was introduced [18, 35] to determine the transaction context for the triggered action with respect to the triggering user-submitted transaction. Basically, the triggered action could be executed *immediately* on behalf of the triggering transaction, *deferred* as a pre-prepare phase of the commit processing, or *detached* in a new and separate transaction. In addition, more advanced couplings are suggested that encompass a dependency between the triggering transaction and a separately spawned transaction for the rule. In particular, the execution of the triggered action may causally depend on the commit or failure of the triggering transaction. As described above, the aDBMSs have been designed on top of a central and monolithic database server. In particular, the event triggering and rule dispatching are integral parts of the aDBMS and are not separated into a generic middleware layer.

In distributed settings, the application process typically spans multiple transactional information systems. Grouping the information access into a single distributed transaction requires resources to be locked for the duration of the transaction, and termination must be coordinated by a 2-phase-commit protocol. While this approach is realized in standardized and commonly applied middleware services [53, 80], the applicability thereof is restricted to tightly coupled systems and thus is not suitable for the integration of autonomous components.



Various extensions to the traditional transaction models have been proposed [40] that relax atomicity and/or isolation in favor of increased concurrency, and in order to preserve the transaction and execution autonomy of involved components. In order to do so, application semantics must be taken into account. A clear separation of concerns between application logic and generic extended transaction services remains an open issue.

Queued transactions [7] support reliable, asynchronous messaging based interactions. Queued transactions are based on the concept of message recovery [9, 97] and provide for the exactly once execution of a request issued from a client application to a transactional information server. Enqueuing/dequeuing of persistent messages is enclosed in a unit of work and is dependent on the overall transaction outcome. Only if the unit of work is committed, will the messages be sent out and consumed. Manipulation of client application state is executed in the same transaction as enqueueing the request and later dequeuing the corresponding reply. The receiver application groups dequeuing of request message, service execution, and enqueueing of reply into a single unit of work. Both client and server are loosely coupled in terms of time-dependency and transaction autonomy, as they do not share a synchronous communication state or a single transaction context.

In transactional publish/subscribe [25], push-style event notifications and database transactions are integrated, separating this active functionality in a middleware layer. In [94] the authors identify the restrictions of queued transactions in common message-oriented middleware and suggest message delivery and message processing transactions. Middleware-mediated transactions [65] (MMT) suggest a synthesis of queued transactions, transactional publish/subscribe, and distributed object transactions, including database transactions. MMT are the basis for integrating notifications and transactions, as we will discuss in Sect. 3.5.

## 2.6 Software Engineering/Management

In publish/subscribe systems the control flow is not explicitly coded. This results in the desirable loose coupling and scalability but makes system management more difficult. Adding or deleting producers of notifications, for example, affects the system's overall functionality. Therefore, notification services can be differentiated by their ability to structure sets of producers and consumers. Operational controls and management tasks can then be bound to these structures.

The channels of the CORBA notification service [75] can be interconnected and managed in event management domains [76]. The domains provide a uniform management interface but do not offer any filtering of notifications between coupled channels. However, producers must still explicitly publish into a specific channel, moving information about application structure into the components and limiting system evolution, a problem which is only recently addressed by reflective middleware [34].

As described in Sect. 2.1, subject-based addressing schemes use a predefined tree of subjects to classify, partition, and select notifications [79]. The subject tree can only be defined according to a specific viewpoint, be it notification content, or network or application layout, thereby making the integration of systems difficult [57]. Many

commercial systems exist (e.g. from TIBCO, IBM, etc.) that extend the basic features with bridges connecting multiple busses, transactional processing, and security. However, the management of notification services is separated from the application functionality, which is affected only implicitly, obscuring the interdependencies and complicating management of the overall system. The Java Message Service [56] does not offer a management interface beyond selecting the persistence and time-to-live of notifications.

The SIENA event notification service is a popular example of a distributed service utilizing content-based filtering [21]. As for all other content-based filtering approaches, the filters may be used to realize visibility constraints, but these issues are not explicitly addressed. The idea of defining event zones is present in several prototypes [54, 78] to limit the distribution of notifications. None of the approaches focuses on visibility control as the central mechanism to coordinate applications and to localize management tasks, which is a known concept in other areas and is employed in DREAM [43].

Peer-to-peer systems have developed interesting strategies in environments without central control. These strategies allow them to be self-organized, maintain a well-defined network topology despite frequent node failures, and offer bounded delivery depth and load sharing. As was mentioned before, several recent efforts [22, 86, 95] try to combine publish/subscribe and peer-to-peer to complement the communication efficiency of the first with the manageability of the latter.

In classical software engineering, event-based interaction is also known. The observer pattern [47] directly follows this approach, and events are used in graphical user interfaces and in software integration and composition. Sullivan and Notkin introduce mediators [93] as a design approach to explicitly instantiate and express integration relationships. An implicit invocation abstraction is used to bundle components and mediators and, with their own interfaces, to compose new components. The Field environment [87] is an early work on tool integration that relies on a centralized server to distribute events. The original approach realizes content-based filtering in a flat space of notifications, which was later extended with the Field Policy Tool by introducing a (manual) mapping of any sent message to a set of message–receiver pairs.

### 3 Architecture

The DREAM middleware platform consists of two main portions: the event notification subsystem and the reactive functionality service. These two subsystems are complemented by the mechanisms for transaction support and scopes, an orthogonal management support. Figure 1 describes schematically the architecture.

Events are produced by sensors or applications. Event producers may be either actively pushing events or be wrapped by event detectors, which pull sensor data or query the application for relevant state changes. Events can be consumed either by a reactive application, i.e. an application that integrates the reactive functionality, or by the reactive functionality service that handles events on behalf of passive applications

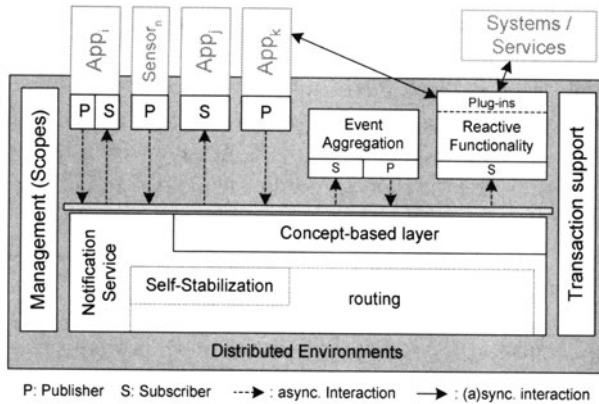


Fig. 1. DREAM architecture

and invokes them as needed. In addition, the reactive functionality service can invoke, via plug-ins, external systems or services (see Sect. 3.4).

Event producers publish event notifications. An event is the observation of a happening of interest. A notification is the reification of an event and includes the event's identity, a timestamp, and the time-to-live for an event. An event notification also includes the parameters of the event, i.e. data, and in the case of heterogeneous environments, the event's interpretation context. Additionally, events carry a transaction context if the notification service is enhanced by transaction support (see Sect. 3.2 for details of the event representation).

The notification service is the delivery mechanism responsible for matching published event notifications with subscriptions. It routes events from their sources to their destinations. The routing mechanism may be realized at the physical level (IP addresses), at the level of subject hierarchies, on the basis of message content if all parties belong to one homogeneous context, or on the basis of concepts, i.e. the terms of an ontology, if publishers and subscribers belong to heterogeneous contexts. The DREAM architecture allows you to bypass a layer if it is not needed.

Publish/subscribe systems work on the basis of filters. Filters reduce the number of notifications by forwarding only those for which subscriptions exist. As such, filters typically do not combine events. In many applications, however, added value can be derived from aggregating events and reacting to a combination of primitive events. The aggregated event should then be forwarded to the subscriber (see Sect. 3.3 for a discussion of event aggregation). Although aggregation could be viewed as being part of the notification mechanism, in DREAM we have realized event aggregation outside the notification mechanism. Event aggregators are treated like any other event-consuming application that can subscribe to events, aggregate them, and publish a new aggregated event. In this way we can accommodate different kinds of event aggregation, e.g. based on event graphs or on streaming queries. Details of

the notification service, as well as the optimizations for filter placement and how to achieve fault-tolerance via self-stabilization, are given in Sect. 3.1.

In publish/subscribe systems the quality of service of the delivery mechanism is not compatible with traditional notions of transactions. Transactional behavior is usually limited to guaranteed delivery of the notification from the publisher to the broker but does not include the delivery to the subscriber. In DREAM we developed a transaction mechanism that allows the client, i.e. the subscriber, to determine the transactional quality of an interaction. Details of how notifications and transactions interact are given in Sect. 3.5.

Finally, the DREAM architecture includes a component that supports modeling and management of event-based systems. The notion of scopes is used to structure the applications and to manage policies bound to this structure. Policies may determine the range of visibility or relevance of events, adapt the notification service functionality, and may be used for the enforcement of security policies. Scopes are described in Sect. 3.6.

### 3.1 Notification Service

The foundation of the DREAM architecture is a delivery mechanism that spans the underlying distributed system. It consists of a distributed event notification service to which applications and other system services are connected as clients. These clients act as producers and consumers of notifications. The notification service itself is an overlay network in the underlying system, consisting of a subset of nodes connected in a network of event brokers. The brokers receive notifications, and filter and forward them in order to deliver published notifications to all attached consumers having a matching subscription. This well-known architecture is used in a number of existing systems. However, we studied the influence of the routing strategy on the scalability and performance of the notification mechanism, specified its characteristics formally to analyze the routing, and extended its dependability with self-stabilization.

**Routing.** We assume content-based routing to develop the routing strategies. Flooding is the simplest approach to implement routing: brokers simply forward notifications to all neighboring brokers and only those brokers to which clients are connected test on matching subscriptions. Flooding guarantees that notifications will reach their destination, but many unnecessary messages (i.e. notifications that do not have consumers) are exchanged among brokers. The main advantage of flooding is its simplicity and that subscriptions become effective instantly since every notification is processed by every broker anyway.

An alternative to flooding is filter-based routing. It depends on routing tables that are maintained in the brokers and contain link-subscription pairs specifying in which direction matching subscriptions have to be forwarded. The table entries are updated by sending new and canceled subscriptions through the broker network. Different flavors of filter-based routing exist. Simple routing assumes that each broker has global knowledge about all active subscriptions. It minimizes the amount of notification traffic, but the routing tables may grow excessively. Moreover, every (un)subscription

has to be processed by every broker, resulting in a high filter forwarding overhead if subscriptions change frequently.

Our experiences have shown that in large-scale systems, more advanced content-based routing algorithms must be applied [73]. Those algorithms exploit commonalities among subscriptions in order to reduce routing table sizes message overhead. We have investigated three of them: identity-based routing, covering-based routing [21], and merging-based routing [72]. Identity-based routing avoids forwarding of subscriptions that match identical sets of notifications. Covering-based routing avoids forwarding of those subscriptions that only accept a subset of notifications matched by a formerly forwarded subscription. Note that this implies that it might be necessary to forward some of the covered subscriptions along with the unsubscription if a subscription is canceled. Merging-based routing goes even further. In this case, each broker can merge existing routing entries to a broader subscription, i.e. the broker creates new covers. We have implemented a merging-based algorithm on top of covering-based routing [71]. In this algorithm, each broker can replace routing entries that refer to the same destination by a single one whose filter covers all filters of the merged routing entries. The merged entries are then removed from the routing table, and the generated merger is added instead and forwarded like a normal subscription. Similarly, the merger is removed if there is an unsubscription for a constituting part of the merged filter or if a subscription arrives that covers a part or the whole merger.

Advertisements can be used as an additional mechanism to further optimize content-based routing. They are filters that are issued (and canceled) by producers to indicate (and revoke) their intention to publish certain kinds of notifications. If advertisements are used, it is sufficient to forward subscriptions only into those subnets of the broker network in which a producer has issued an overlapping advertisement, i.e. where matching notifications can be produced. If a new advertisement is issued, overlapping subscriptions are forwarded appropriately. Similarly, if an advertisement is revoked, it is forwarded and remote subscriptions that can no longer be serviced are dropped. Advertisements can be combined with all routing algorithms discussed above.

**Self-stabilization of broker networks.** Being able to formally reason about the correctness of a system is an often-neglected prerequisite for avoiding bugs and misconceptions. For DREAM a formalization of the desired system semantics is used to evaluate the routing strategies and their implementation [45, 71]. Based on these formal semantics it was also easier to incorporate and validate a very strong fault-tolerance property into the notification service: self-stabilization.

Self-stabilization, as introduced by Dijkstra [37], states that a system being in an arbitrary state is guaranteed to eventually arrive at a legitimate state, i.e. a state starting from which it offers its service correctly. The arbitrary initial state models the effect of an arbitrary transient fault. Therefore, self-stabilization is generally regarded as a very strong fault-tolerance property. A self-stabilizing publish/subscribe system is guaranteed to resatisfy its specification in a bounded number of steps as long as the broker topology remains connected and the programs of the surviving brokers are not corrupted.

Self-stabilization in the broker network is realized by discarding broken and out-dated information about neighbors. This is accomplished by the use of leases. A node must renew its subscription to maintain it in the network. Since the process of lease renewal is idempotent, multiple renewals do not affect the functioning of the system and it is enough to renew the lease any time before it expires. A lease renewal just updates the timestamp for the lease expiration time that is stored by the brokers with each entry. If filter merging is performed, the resulting routing entries must be renewed, too. To clean up the effects of internal transient faults, brokers validate their routing tables periodically to remove entries with expired leases. Transient failures of network links are not masked by this approach, and lost notifications are not automatically retransmitted. However, the approach using leases guarantees that transient faults do not upset system operation longer than necessary. In this sense, the system infrastructure can be regarded as *self-healing*.

The timing conditions for lease renewal depend on the link delay and the network diameter, i.e. the time needed to process and forward a subscription message and the number of links that must be traversed, respectively. The leasing period is the time for which a lease is granted, and determines the stabilization time needed to recover from an error. There is an obvious trade-off between these values in that more accurate behavior demands more control messages and increases the overhead. The system has been designed to adapt leasing periods to changing network characteristics. For this, the self-stabilization methodology can be applied again.

### 3.2 Concept-Based Layer

To express subscriptions, consumers need to know about the content of the events and messages that are being exchanged. This means that consumers must know details about the representation and assumed semantics of message content. Notification services at best specify message content by means of Interface Definition Language or no explicit specification is made at all. Thus, in the best case, only the data structure of the notification content is specified while leaving required information about data semantics implicit. This reflects a low level of support for event consumers that based on this scarce information must express their interest (subscriptions) without having a concrete definition of the meaning of messages assumed by their producers. Without this kind of information event producers and consumers are expected to fully comply with implicit assumptions made by participants.

The approach taken here tries to solve this problem by providing a concept-based layer on top of the delivery mechanism. This layer provides a higher level of abstraction in order to express subscription patterns and to publish events with the necessary information to support their correct interpretation outside the producers' boundaries.

**Representing events.** Events are represented using the MIX model (*Metadata based Integration model for data eXchange*) [11, 12]. MIX is a self-describing data model since information about the structure and semantics of the data is given as part of the data itself. It refers to concepts from a domain-specific ontology to enable the

semantically correct interpretation of event content, and supports an explicit description of the underlying interpretation context in the form of additional metadata. In other words, the underlying ontology defines the set of concepts that are available to describe data and metadata from a given domain.

In our infrastructure ontologies are organized in three categories [28]:

- The *basic representation ontology* contains the basic numeric and character data types, URL, currency, date, time, and physical dimensions. The representation ontology contains the necessary definitions for (un)marshaling.
- The *infrastructure-specific ontology* contains events (primitive, temporal, composite, etc.), notifications, and policies, such as consumption modes or lifetimes of events and notifications. These are the basic concepts of an event-based distributed infrastructure.
- The *domain-specific ontologies* contain the concepts for the various application domains, for example, auctions, car-specific services, or air traffic control. Some detail on these application domains is given in Sect. 4.

Events from heterogeneous sources can be integrated by converting them to the target context required by the respective consumer. This can be done by applying conversion functions that may include the calculation of a simple arithmetic function, a database access, or the invocation of an external service. Conversion functions can be specified in the underlying ontology if they are domain-specific and application-independent. Application-specific or service-specific conversion functions may be defined and stored in an application-specific conversion library and will overwrite those given in the ontology.

**Concept-based addressing.** Since events are represented with concepts of the ontology, we can provide a high-level subscription specification. Consumers define their subscription patterns by also referring to the underlying ontology, which we call *concept-based addressing*. Subscriptions include the possibility of expressing consumers' interests using local conventions for currency, date-time format, system of units, etc. In this way, consumers do not need to take care of proprietary representations and all participants use a common set of concepts to express their interests. Moreover, event consumers are allowed to express their interests without previously knowing the assumptions made by event producers. This means that one producer can send events that contain prices expressed in USD while other producers can express prices in EUR. Consumers of these events simply need to specify at subscription time the context (in this case, the currency) in which they want to get the content of events. This is done by automatically applying the respective conversion functions to events before they are passed to consumers. Additionally, event consumers can benefit from the ontology and their relationships, e.g. generalization/specialization. They can use abstract concepts for specifying their subscriptions and receive instances of specializations of the abstract concept.

### 3.3 Event Aggregation

Events can be either primitive or composite. In most practical situations primitive events, e.g. events detected by basic sensors or produced by applications, must be combined. Usually, this composition or aggregation relies on an event algebra that may include operators for sequence, disjunction, conjunction, etc. Existing event algebras were developed for centralized systems and depend on the ability to determine the order of occurrence of events [101].

However, these event algebras and consumption policies depend on a total order of events and are based on point-based timestamps of a single central clock. These assumptions are invalidated by the inherent characteristics of distributed and heterogeneous environments. Exact knowledge of event occurrence as a point on the timeline ignores the effects of granularity and indeterminacy of time instants. As a matter of fact, granularity and indeterminacy of event occurrence time are two sides of the same coin. The timestamp granularity might reflect the inaccuracy of clocks and time measurements [77], as well as the inaccuracy of event observation – think of a polling event detector running once an hour. A coarse granular event occurrence time might be chosen on purpose, because in the universe of discourse a finer granularity is not appropriate. For example, the event could be associated with an activity that itself has a duration but this duration is not of interest. In that case, the atomicity of an event is the result of an abstraction. As another example, consider planning and scheduling of activities, which is best modeled at an appropriate time granularity related to the heartbeat of the process. In typical supply chain management (SCM) scenarios for example, events are recorded at the granularity of calendar day, and only in some cases down to hours and minutes.

In all cases, the event occurrence time must be considered to be indeterminate to some extent. As a consequence, time indeterminacy must be reflected in the time model and explicitly recognized and reported when composing events in distributed and heterogeneous systems. Three factors are crucial in a generic event composition service: (1) the proper interpretation of time, (2) the adoption of partial order of events, and (3) the consideration of transmission delays between producers and consumers of events.

The first point is basically related to timestamping events. Here, timestamps are represented with accuracy intervals with reliable error bounds that reflect the indeterminacy of occurrence time, imposed by timestamp granularity and inaccuracy of time measurements [63]. In our infrastructure an abstract timestamp concept is defined and particular timestamp representations can be specialized for different scenarios and environments according to the adopted time model.

In regard to the second point, we adopted a partial order of events. Consequently, correlation methods include the possibility of throwing an exception (e.g. `CannotDecide`) in order to announce such an uncertainty when comparing events. In this way, the underlying infrastructure is responsible for announcing an ambiguous situation to a higher level of decision, allowing the use of application semantics for the resolution [28]. In this approach, user-defined or predefined policies can be configured in order to handle these situations.



Third and finally, we handle transmission delays and network failures by using a combination of a window scheme with a heartbeat protocol. The window mechanism is used to separate the history of events into *stable past* and *unstable past and present* that are still subject to change. For composition purposes only events in the stable past are considered. With all this, it can be guaranteed in all cases that: (a) situations of uncertain timestamp order are detected and the action taken is exposed and well defined, and (b) events are not erroneously ordered.

The infrastructure of the event aggregation service is based on the principles of components and containers. Containers control the event aggregation process, while components define the event operator logic. As mentioned before, the aggregation service is treated like any other event consumer that can subscribe to events; it aggregates them and finally publishes the aggregated event. The handling of time indeterminacy and network delays are encapsulated in such a container.

### 3.4 Reactive Functionality

An event-based system depends on the appropriate reaction to events. Because we are targeting open environments, the reactive components must be able to interpret events originating from heterogeneous sources. Therefore, the context information presented in Sect. 3.2 is not only needed for event composition and notification, but also for interpretation of events and their parameters by the components reacting to them.

In DREAM the traditional processing of event-condition-action (ECA) rules is decomposed into its elementary and autonomous parts [29]. These parts are responsible for event aggregation, condition evaluation, and action execution. The processing of rules is then realized as a composition of these elementary services on a per-rule basis. This composition forms a chain of services that are in charge of processing the rule in question. These elementary services interact among them based on the notification service. As mentioned before, the reactive service is treated like any other event consumer that can subscribe to events. When events of interest (i.e. those that trigger rules) are notified, the corresponding rule processing chain is automatically activated. Elementary services (i.e. action execution) that interact with external systems or services use *plug-ins* for this purpose. Besides that, plug-ins are responsible for maintaining the semantic target context of the system they interact with, making possible the meaningful exchange of data.

On the foundations of an ontology-based infrastructure, a reactive functionality service was developed that provides the following benefits: services interact using an appropriate vocabulary at a semantic level, events from different sources are signaled using common terms and additional contextual information, and rule definition languages can be tailored for different domains using a conceptual representation, providing end-users the most appropriate way to define rules. This conceptual representation enables the use of a ‘generic’ reactive functionality service for different domains, making the underlying service independent from the rule specification. For instance, this service is used in the context of online meta-auctions as well as the

Internet-enabled car (see Sect. 4). Details related to the reactive functionality service can be found in [28].

### 3.5 Notifications and Transactions

In the event-based architectural style the event producer is decoupled from the event consumer through the mediator. Therefore, any transaction concept in an event-based system must include the mediator. On the other hand, applications will be implemented in some (object-oriented) programming language. Object transactions, as provided, for example, by CORBA in the OTS, are based on a synchronous, one-to-one request/reply interaction model that introduces a tight coupling among components. The challenge is therefore to combine notifications with conventional transactional object requests [64] into middleware-mediated transactions (MMTs) [65]. MMTs extend the atomicity sphere of transactional object requests to include mediators and/or final recipients of notifications. To properly understand MMTs and their benefits, we briefly describe below the basic issues of component connection, interaction, and reliability.

The **topology** of interacting components may be *fixed* or *variable* and could be 1:1, 1:n, or n:m. Depending on the knowledge of the counterpart, the **binding** of producer to consumer may be *reference-based* (at least one party has an exact reference to the other) or *mediator-based* (parties have no direct reference to each other but interact through the mediator based on subjects, topics, content, or concepts).

We may consider the **life-cycle** dependencies between interacting components. The interaction model may not require the components to be available at the same time. This means that components can execute as *time-independent*. If components must be available at the same time in order to interact, then they are *time-dependent*. Time-independence is a major aspect of loose coupling in addition to mediator-based interactions.

The **synchronicity** of interactions describes the synchronization between components and whether they block while waiting for completion of the interaction. We distinguish *synchronous*, *asynchronous*, or *deferred synchronous* behavior.

The **delivery** guarantees for notifications may be *best-effort*, *at-most-once*, *at-least-once* or *exactly-once*. The first two are unreliable, while the last two are reliable. In addition to delivery we must consider the coupling to the execution of reactions. The **processing** of the subscriber may be coupled *best effort* or *atomic transactionally coupled*. In case there is a coupling between execution of producer, subscriber, and mediator, the **recovery** from situations of failures could be *forward* (outgoing) or *backward* (incoming) from the point of view of the publishing transaction.

To illustrate the differences among well-known mechanisms we consider object-oriented communication and transactions and traditional publish/subscribe. For common remote method invocation styles, such as in CORBA and J2EE, the topology is fixed to 1:1 connections and bindings are reference-based. Also, components are time-dependent on each other and invocations are typically synchronous. As there is no mediator, transactions either group the initiator (client) and the responder (server)

of interactions into a single atomicity sphere, or the execution of initiator and responder is carried out in independent transaction contexts. There is no middleware support for message-based recovery at the client. As a consequence, forward recovery after client failure must be dealt with at the application level, and guaranteeing exactly-once execution – from the client’s point of view – is a tedious task.

Traditional publish/subscribe systems provide a more flexible and loose coupling, as they support variable  $n:m$  topologies with time-independent interactions via a mediator. However, transactional delivery is restricted to the mediator, and subscribers are restricted to transactional consumption of the notifications. There is no means for transactionally coupling with respect to the contexts of publisher and subscriber.

In order to integrate producers, mediators, and subscribers, a more flexible transactional framework is needed. This framework must provide the means to couple the visibility of event notifications to the boundaries of transaction spheres and the success or failure of (parts of) a transaction. It must also describe the transactional context in which the consumer should execute its actions. It must specify the dependencies between the triggering and the triggered transactions, dynamically spanning a tree of interdependent transactional activities.

Table 2 illustrates the possible options for coupling event producers, mediators, and recipients of notifications. With *immediate visibility*, events are visible to consumers as soon as they arrive at the consumer’s site and are independent of the outcome of the triggering transaction. *On commit (on abort) visibility* specifies that a consumer may only be notified of the event if the transaction has committed (aborted). *Deferred visibility* requires that the consumer be notified as soon as the producer starts commit processing.

	Modes
Visibility	Immediate, on commit, on abort, deferred
Transaction context	None, shared, separate
Forward dependency	None, commit, abort
Backward dependency	None, vital, mark-rollback
Production	Transactional, independent
Consumption	On delivery, on return, atomic, explicit

**Table 2.** Transaction framework provided couplings

A commit (abort) *forward dependency* specifies that the triggered reaction only commits if the triggering transaction commits (aborts). *Abort forward dependencies* are a powerful concept to realize compensations and to enable recovery of long running processes.

A *backward dependency* constrains the commit of the triggering transaction. If the reaction is *vital* coupled, the triggering transaction may only commit if the

triggered transaction was executed successfully. If the consumer is coupled in *mark-rollback* mode, the triggering transaction is independent of the triggered transaction's commit/abort, but the consumer may explicitly mark the producer's transaction as rollback-only. Both backward dependencies imply that a failure to deliver the event will cause the triggering transaction to abort.

If the reaction is coupled in *shared mode*, it will execute on behalf of the triggering transaction. Of course, this implies a forward and a backward dependency, which is just the semantics of spheres of atomicity. Otherwise, the reaction is executed in its own atomicity sphere, i.e. *separate top* and the commit/abort dependencies to the triggering transaction can be established as described above.

Once an event has been consumed, the notification is considered as delivered and will not be replayed in case the consumer crashes and subsequently restarts. The consumption modes allow a loose or tight coupling of event consumption to the consumer's execution context and its transaction boundary. We have implemented the above framework in the X<sup>2</sup>TS prototype [64]. It is discussed in Sect. 4 with the other prototypes.

### 3.6 Scopes

Despite the numerous advantages offered by the loose coupling of event-based interaction, a number of drawbacks arise from the new degrees of freedom. Event systems are characterized by a flat design space in which subscriptions are matched against all published notifications without discriminating producers. This makes event systems difficult to manage. A generic mechanism is needed to control the visibility of events, e.g. for security reasons and for structuring system components, extending visibility beyond the transactional aspects presented in Sect. 3.5.

Scopes [44] allow system engineers to exert explicit control on the event-based interaction; it is a functionality orthogonal to the transaction mechanism that stretches across the different layers of the DREAM architecture (Fig. 1). The following examples illustrate the kinds of explicit control offered. In order to overcome the flat design space of event systems, scopes offer interfaces that limit the distribution of published notifications. A scope bundles several producers and consumers, and it limits the visibility of events in the sense that notifications are delivered to consumers within the same scope but are a priori invisible elsewhere. According to an assigned interface a scope may forward internal notifications to the outside, and vice versa. It encapsulates the composed producers and consumers and may itself be a member of other scopes, thus creating new clients of the notification service as long as the resulting graph is acyclic [45]. The achieved information hiding may be used to abstract from details of application structure.

Ontologies are used to map notifications between producers and consumers of different application contexts. However, context is typically not identified with arbitrary single producers or consumers, but instead characterizes different (parts of) applications that are to interact at run time. Context descriptions can be bound to scopes, which bundle coherent groups of cooperating clients and share a common

application context. For example, a scope could restrict visibility to trade-event notifications originating from NASDAQ. The scope could have an associated context 'US-exchange' that defines the currency, the valid fractions in a quote, the date and time formats, etc.

Scopes are also used to exert a finer control of who is going to receive a given notification. A typical example for such a delivery policy is a 1-of- $n$  delivery to only one out of a set of possible consumers. Note that, from the producer's and the consumer's points of view, event-based semantics are maintained, and only the interaction within a specific scope is to be refined. Delivery policies bound to a scope specify the refined semantics of delivery in this scope.

Different aspects of controlling visibility of events are addressed in a number of existing products and research contributions, ranging from administrative domains [55] to full database access control [82]. However, prior to proposing scopes we were missing an approach that is orthogonal to the other aspects of DREAM. We see scopes as the means by which system administrators and application developers can configure an event-based system. Scopes offer an abstraction to identify structure and to bind organization and control of routing algorithms, heterogeneity support, and transactional behavior to the application structure. They delimit application functionality and contexts, controlling side effects and associating ontologies at well-defined points in the system. This is of particular importance as platforms of the future must be configurable not only at deployment time but also once an application is in operation.

## 4 Proof-of-Concept Systems

In this section we present prototypical implementations that illustrate the concepts discussed above.

### 4.1 Rebeca

REBECA (Rebeca Event-Based Electronic Commerce Architecture) [45, 70] is a content-based notification service that implements the event notification and routing described above and does not rely on only a single routing algorithm, but implements all the algorithms presented in Sect. 3.1. Scopes are implemented on top of REBECA. Several options exist that differ in the degree of distribution, flexibility to evolve, and the necessary management overhead. A completely distributed solution divides the existing routing tables into scope-specific tables and uses the plain routing mechanisms for intrascope delivery. Cross-scope forwarding and the application of interfaces and context mappings are handled when notifications are transferred between routing tables. A number of alternatives are available here: scope crossing may be centralized to enforce some security policy, to audit traffic, or to bridge specialized implementations of notification delivery. On the other hand, single scopes may be realized by a centralized notification broker, offering a finer control of notification forwarding and delivery policies.

Measurements of the characteristics of the routing algorithms were conducted with the help of a stock trading application that processed real data feeds taken from the Frankfurt stock exchange [73]. The results clearly show the beneficial effects of using advanced routing algorithms in the REBECA notification service.

## 4.2 X<sup>2</sup>TS

X<sup>2</sup>TS integrates distributed object transactions with publish/subscribe systems [64] and is a prototype for the MMT concept in DREAM. X<sup>2</sup>TS supports the full range of object transaction features, such as indirect context management, implicit context propagation, and interposition for and integration with X/Open XA resources such as RDBMS and MOM resource managers. The prototype supports a push-based interface with one-at-a-time notifications. It assumes that events are instances of types and that subscriptions may refer to specific types and to patterns of events. Subscription to patterns of events and coupling modes are imposed by the consumer by configuring the service proxy.

The X<sup>2</sup>TS prototype implementation currently supports the following middleware-mediated transaction (MMT) features:

- Deferred, on abort and on commit visibility.
- Checked transactions to support forward couplings and vital backward dependencies.
- Backward processing dependencies with the possibility to define groups of subscribers to be vital.
- Production policy is nontransactional but can be explicitly programmed as transactional.
- Consumption policies are atomic, on delivery, on return, explicit.
- X<sup>2</sup>TS manages transactions at the consumer side. The consumer may provide its own transaction context, select a shared context or let X<sup>2</sup>TS create a new transaction for each notification. Grouping of notifications in one transaction is possible through the definition of a composite event.

Reliable, exactly-once delivery and recoverable processing of events is realized in a straightforward combination of *producer at-least-once* and *consumer-at-most-once* strategy. The basic idea is as follows. The consumer site uses a persistent and transactional log, which keeps track of the identifiers of processed notifications. Arriving notifications are only delivered if the identifier is not found in the log. Logging the identifier is atomically executed as part of the consumer's unit of work, and on successful completion thereof an acknowledgment is returned to the publisher. The publisher keeps on resending notifications until the transmission is acknowledged by each subscriber. X<sup>2</sup>TS allows subscribers to the same event to specify different couplings. While one subscriber may react on commit of the triggering transaction, the other may need to react as soon as possible.

The realization of visibilities and dependencies is basically achieved by sending events about transaction status changes, and the propagation of the publisher's transaction context within the notification message. Thereby, events need only to be published

once. At the consumer site, a built-in event composition enforces the visibilities and dependencies. Detection of event patterns is realized through pluggable event compositors. The application of these concepts to multilevel transaction services, which encompass activities at the process level, is considered to be straightforward.

### 4.3 Meta-Auctions

Auctions are a popular trading mechanism. The advent of auction sites on the Internet, such as eBay or Yahoo, has popularized the auction paradigm and has made it accessible to a broad public that can trade practically anything in a consumer to consumer interaction. However, the proliferation of sites makes life more difficult for the buyer. Consider the case of a collector. With the current auction sites, she has to manually search for the item of interest, possibly visiting more than one auction site. If successful, she might end up being engaged in different auctions at multiple auction sites. There are two obvious shortcomings to this approach: first, the user must poll for new information and might miss the window of opportunity, and second, the user must handle different auction sites with different category setups and different handlings. This motivates the need for the meta-auction broker [13], which provides a unified view of different auction sites and services for category browsing, item search, auction participation and tracking (Fig. 2).

Notifications about events, such as the placement of a highest bid, and their timely delivery represent valuable information. Propagation of events leads to a useful and efficient nonpolling realization of an auction tracking service. Therefore, publish/-subscribe as an additional interaction paradigm is needed for disseminating process-related information efficiently. Events related to the auction process are disseminated using the concept-based notification service presented in this paper. This way, publishers and subscribers use a common semantic level of subscription.

Each site participating in the meta-auction system provides information about items and the auction process, but does not share a global data schema nor may we assume a global schema for notifications. Today, the exact meaning of terms, entities, and notifications used by different auction sites is still left implicit. To enable the brokering between different participating auction sites, the precise understanding of the terms used by each site is needed and should be made explicit through a domain-specific ontology. We introduced an ontology-based infrastructure for explicit metadata management on top of which the meta-auction service can be realized and the mappings through which the domain-specific ontologies can be combined.

The auction process itself can be defined using state charts. Since they are event-driven, they can be easily implemented with ECA rules. Different sets of rules can describe different types of auction processes (ascending, reverse, dutch, etc.) [30].

To track an item of interest during an auction process, e.g. to detect that another bidder has reached a highest bid or that the deadline of an auction is approaching, an agent can be used. Bidders can benefit from the reactive service to program their own agents. In contrast to current agent bidders that are owned, controlled, and implemented by the auction house, these agents can react to happenings of the auction process according to the bidders' strategy.

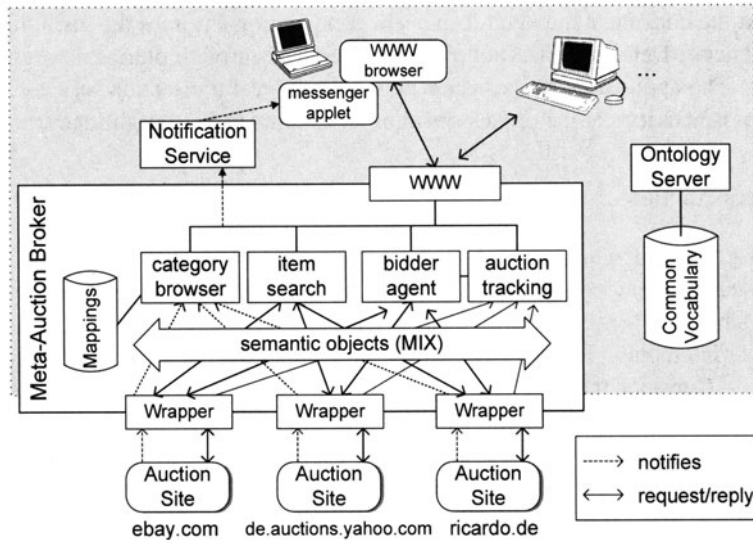


Fig. 2. Meta-auction architecture

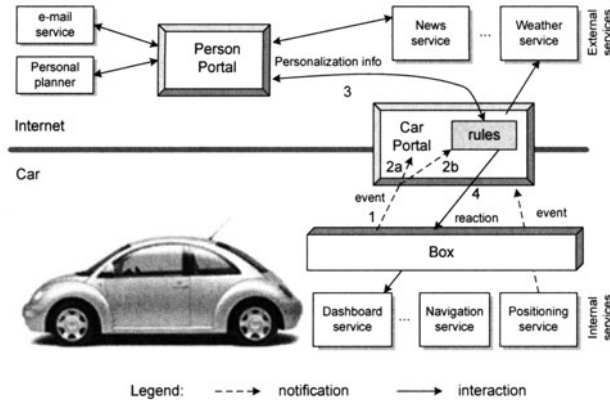
#### 4.4 Internet-Enabled Car

Similar to other pervasive computing environments, cars will see a convergence of Internet, multimedia, wireless connectivity, consumer devices, and automotive electronics. Wireless links to the outside world open up a wide range of telematics applications. Automotive systems are no longer limited to information located on-board, but can benefit from a remote network and service infrastructure.

Consider the scenario where vehicles, persons, and devices have a Web presence (or portal). Within this scenario new possibilities emerge, e.g. the adjustment of instruments according to personal preferences, favorite news channels, sports, music, or access to one's e-mail and calendar through the portals. Through the portals this can be made independent of a particular car and could be applied to any rental car. But not only instruments can be adjusted, services can be personalized too. Services such as "find and set the route to the next gas station", or "book an appointment to change oil" can take into account a company's, and/or a driver's preferences. The portals are enhanced with the reactive functionality service in order to react to events of interest according to user preferences. Preferences are stored and managed by the portal manager.

Vehicles are equipped with a GPS receiver and a box. This box plays the role of a mediator between the vehicle itself and the external world. It can access a vehicle's electronic and diagnostic interfaces (like interfaces J1850, ODB-II), and it is responsible for announcing status changes to its portal. The portal manager can react to them by using the reactive functionality service (Fig. 3).





**Fig. 3.** An Internet-enabled car

A prototype [31] was developed in conjunction with industry to show the reaction of a personalized car to different situations based on a set of user-defined rules. Implemented services include:

*Adjustment of instruments:* When the driver gets into the car all her preferences (preferred language, units used for temperature, distance and velocity, format of date/time, radio stations, music preferences, etc.) are automatically loaded. The driver's identity is detected (e.g. smartkey) when getting into the car, and the car's box communicates this event to its portal. As a reaction, a rule is fired that reads the driver's preferences and contacts the box to set/load them into the car's instruments.

*Low fuel:* A sensor signals this event, and a location service is invoked to find the next gas station, considering current geographical position, destination, and the preferred vendor.

*Driving to work:* A commuter gets into the car on a workday and the current time is between 8:00 am–9:00 am. As a reaction the best route to work avoiding traffic jams is offered and passed to the navigation service; today's scheduled meetings are reviewed; company news and other personalized news are obtained; and e-mails can be read. Because drivers should concentrate on driving, all this information is read out by using a text-to-speech service.

## 5 Conclusions and Future Work

We have shown the main properties that we expect a reactive event-based middleware platform to exhibit:

- event detection and composition/aggregation mechanisms in a highly distributed environment
- robust and scalable event notification that exploits a variety of filter placement and self-stabilization strategies

- content-based notification to formulate powerful filters
- concept-based notification to extend content-based filtering to heterogeneous environments
- middleware-mediated transactions that integrate notifications and transactions with well-defined semantics for visibility, life-cycle dependencies, synchronicity, delivery, and recovery dependencies
- configuration and administration primitives, such as scopes, for both deployment and run-time configurability, as well as attachment and administration of policies.

The complete set of requirements for the wide spectrum of potential applications cannot be anticipated. In addition, experience has shown that generic platforms can only cover a certain percentage of the business semantics. Therefore, we believe that two conditions must be clearly fulfilled by the DREAM platform. First, no false claims should be made. That is, if the semantics cannot be clearly defined, the middleware should not give a false sense of security to the user, and rather let the application developer compensate based on application semantics. Second, the configurability of event-based platforms must be guaranteed.

We have developed both the necessary concepts and prototype solutions for individual parts of such a system. The results have been published individually. In this paper we presented a system overview that describes how the parts interact. We are in the process of reengineering the platform to eliminate the gaps and inconsistencies that are typical when integrating several theses.

## References

1. S. Abiteboul, S. Cluet, and T. Milo. Correspondence and translation for heterogeneous data. In *Proc. Int'l. Conf. on Database Theory (ICDT)*, 1997.
2. S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In *Proc. ACM Int'l Conference on Management of Data (SIGMOD)*, 1995.
3. G. Alonso, C. Hagen, and A. Lazcano. Processes in electronic commerce. In *Proc. ICDCS Workshop on Electronic Commerce and Web-Based Applications (ICDCS 99)*, May 1999.
4. J. Bailey, A. Poullovassilis, and P.T. Wood. Analysis and optimisation of event-condition-action rules on XML. *Computer Networks*, 39(3):239–259, 2002.
5. H. Balakrishnan, D. Carney, U. Cetintemel, M. Cherniack, M. Stonebraker, Y. Xing, and S. Zdonik. Aurora: A distributed stream processing system. In *Proc. Int'l Conference on Innovative Data Systems Research (CIDR)*, 2003.
6. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. In *Scientific American*, May 2001.
7. P. A. Bernstein and E. Newcomer. *Principles of Transaction Processing for Systems Professionals*. Morgan Kaufmann, 1996.
8. P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA, 1987.
9. P.A. Bernstein, M. Hsu, and B. Mann. Implementing recoverable requests using queues. In H. Garcia-Molina and H. V. Jagadish, editors, *Proc. ACM Int'l Conference on Management of Data (SIGMOD)*, pages 112–122. ACM Press, 1990.

10. A. Bonifati, S. Ceri, and S. Paraboschi. Active rules for XML: A new paradigm for e-services. *The VLDB Journal*, 10(1):39–47, 2001.
11. C. Bornhövd. *Semantic Metadata for the Integration of Heterogeneous Internet Data (in German)*. PhD thesis, Darmstadt University of Technology, 2000.
12. C. Bornhövd and A.P. Buchmann. A prototype for metadata-based integration of internet sources. In *Proc. CAiSE*, volume 1626 of *LNCS*. Springer, 1999.
13. C. Bornhövd, M. Cilia, C. Liebig, and A.P. Buchmann. An infrastructure for meta-auctions. In *Proc. WECWIS'00*, 2000.
14. T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation, W3C, February 1998.
15. D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft, W3C, March 2002. <http://www.w3.org/TR/rdf-schema>.
16. A. Buchmann and C. Liebig. Distributed, object-oriented, active, real-time DBMSs: We want it all – do we need them (at) all? *Annual Reviews in Control*, 25, January 2001.
17. A.P. Buchmann. *Architecture of Active Database Systems*, chapter 2, pages 29–48. In Paton [84], 1999.
18. A.P. Buchmann, A. Deutsch, J. Zimmermann, and M. Higa. The REACH active oodbms. In *Proc. ACM Int'l Conference on Management of Data (SIGMOD)*, page 476. ACM Press, 1995.
19. D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S.B. Zdonik. Monitoring streams—a new class of data management applications. In *Proc. Int'l Conference on Very Large Data Bases (VLDB)*, 2002.
20. A. Carzaniga, D. R. Rosenblum, and A. L. Wolf. Challenges for distributed event services: Scalability vs. expressiveness. In *Engineering Distributed Objects (EDO'99)*, Los Angeles, CA, May 1999.
21. A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.
22. M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8), 2002.
23. S. Chakravarthy, J. Jacob, N. Pandrangi, and A. Sanka. Webvigil: An approach to just-in-time information propagation in large network-centric environments. In *Proc. 2nd Int'l Workshop on Web Dynamics (in conjunction with WWW2002)*, 2002.
24. S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data and Knowledge Engineering*, 14(1):1–26, November 1994.
25. A. Chan. Transactional publish/subscribe: The proactive multicast of database changes. In *Proc. ACM Int'l Conference on Management of Data (SIGMOD)*. ACM Press, June 1998.
26. S. Chandrasekaran et al. Telegraphcq: Continuous dataflow processing for an uncertain world. In *Proc. Int'l Conference on Innovative Data Systems Research (CIDR)*, 2003.
27. S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proc. Int'l Conference on Very Large Data Bases (VLDB)*, pages 606–617, September 1994.
28. M. Cilia. *An Active Functionality Service for Open Distributed Heterogeneous Environments*. PhD thesis, Darmstadt University of Technology, 2002.
29. M. Cilia, C. Bornhövd, and A. P. Buchmann. Moving active functionality from centralized to open distributed heterogeneous environments. In *Proc. Int'l Conference on Cooperative Information Systems (CoopIS)*, volume 2172 of *LNCS*. Springer, 2001.

30. M. Cilia and A.P. Buchmann. An active functionality service for e-business applications. *ACM SIGMOD Record*, 31(1):24–30, 2002.
31. M. Cilia, P. Hasselmeyer, and A.P. Buchmann. Profiling and internet connectivity in automotive environments. In *Proc. Int'l Conference on Very Large Data Bases (VLDB)*, pages 1071–1074, Hong Kong, August 2002. Morgan-Kaufmann.
32. C. Collet. The NODS project: Networked open database services. In K. Dittrich et al., editor, *Object and Databases 2000*, number 1944 in LNCS, pages 153–169. Springer, 2000.
33. D. Conolly, F. van Harmelen, and I. Horrocks et al. DAML+OIL (March 2001) Reference Description. W3C Note, W3C, December 2001.
34. E. Curry, D. Chambers, and G. Lyons. Reflective channel hierarchies. In *Proc. 2nd Workshop on Reflective and Adaptive Middleware, Middleware 2003*. Rio de Janeiro, Brazil, 2003.
35. U. Dayal, B. Blaustein, A.P. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, D.R. McCarthy, A. Rosenthal, S.K. Sarin, M.J. Carey, M. Livny, and R. Jauhari. The HiPAC project: Combining active databases and timing constraints. *ACM SIGMOD Record*, 17(1), March 1988.
36. A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data in relations. In *Proc. Workshop on Query Processing for Semistructured Data and Non-standard Data Formats*, Jerusalem, Israel, 1999.
37. E.W. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.
38. C.E. Dyreson and R.T. Snodgrass. Valid-time indeterminacy. In *Proc. IEEE Int'l Conference on Data Engineering (ICDE)*, 1993.
39. B. Eisenberg and D. Nickull. ebXML Technical Architecture Specification v1.04. Technical report, February 2001. <http://www.ebxml.org>.
40. A.K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992.
41. F. Fabret, F. Llirbat, J. Pereira, A. Jacobsen, K. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe. In *Proc. ACM Int'l Conference on Management of Data (SIGMOD)*, pages 115–126, 2001.
42. D.C. Fallside. XML Schema Part 0: Primer. W3C Recommendation, W3C, May 2001.
43. L. Fiege, M. Mezini, G. Mühl, and A.P. Buchmann. Engineering event-based systems with Scopes. In *Proc. ECOOP'02*, volume 2374 of LNCS. Springer, 2002.
44. L. Fiege, M. Mezini, G. Mühl, and A.P. Buchmann. Engineering event-based systems with Scopes. In *Proc. ECOOP'02*, volume 2374 of LNCS. Springer, 2002.
45. L. Fiege, G. Mühl, and F.C. Gärtner. A modular approach to build structured event-based systems. In *Proc. ACM SAC'02*, 2002.
46. H. Fritschi, S. Gatzju, and K. Dittrich. FRAMBOISE – an approach to framework-based active data management system construction. In *Proc. CIKM'98*, pages 364–370, Maryland, November 1998.
47. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, Reading, MA, USA, 1995.
48. S. Gatzju and K. R. Dittrich. Events in an active object-oriented database system. In *Proc. RIDS'93*, 1993.
49. S. Gatzju, A. Koschel, G. v. Buetzingsloewen, and H. Fritschi. Unbundling active functionality. *ACM SIGMOD Record*, 27(1):35–40, March 1998.
50. N.H. Gehani, H.V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model & implementation. In Li-Yan Yuan, editor, *Proc. Int'l Conference*

- on Very Large Data Bases (VLDB), pages 327–338, Vancouver, Canada, August 1992. Morgan Kaufmann.
51. A. Geppert and D. Tombros. Event-based distributed workflow execution with EVE. In *Proc. IFIP/ACM Int'l Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware)*, The Lake District, England, September 1998.
  52. J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.
  53. The Open Group. Distributed Transaction Processing: Reference Model, Version 3 . Technical Report G504, The Open Group, February 1996.
  54. R. Gruber, B. Krishnamurthy, and E. Panagos. The architecture of the READY event notification service. In *Proc. 19th IEEE Int'l. Conf. on Distributed Computing Systems Middleware Workshop*, Austin, Texas, May 1999. IEEE.
  55. R.E. Gruber, B. Krishnamurthy, and E. Panagos. The architecture of the READY event notification service. In *Proc. Middleware Workshop ICDCS*, 1999.
  56. M. Hapner, R. Burrigge, and R. Sharma. Java Message Service. Specification Version 1.0.2, Sun Microsystems, JavaSoftware, November 1999.
  57. W. Harrison and H. Ossher. Subject-oriented programming (A critique of pure objects). In A. Paepcke, editor, *Proc. 8th ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '93)*, pages 411–428, Washington, DC, USA, 1993.
  58. H. Kopetz. Sparse time versus dense time in distributed real-time systems. In *Proc. ICDCS*, 1992.
  59. A. Koschel and P. Lockemann. Distributed events in active database systems—letting the genie out of the bottle. *Data & Knowledge Engineering*, 25(1-2):29–53, March 1998.
  60. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *CACM*, 21(7):558–565, July 1978.
  61. O. Lassila and R.R. Swick. Resource description framework (rdf) model and syntax specification. W3C Recommendation, W3C, February 1999.
  62. A. Lazcano, G. Alonso, H. Schuldt, and C. Schuler. The WISE approach to electronic commerce. *Int'l Journal of Computer Systems Science and Engineering*, 15(5), 2000.
  63. C. Liebig, M. Cilia, and A.P. Buchmann. Event composition in time-dependent distributed systems. In *Proc. Int'l Conference on Cooperative Information Systems (CoopIS)*, 1999.
  64. C. Liebig, M. Malva, and A.P. Buchmann. Integrating notifications and transactions: Concepts and X<sup>2</sup>TS prototype. In *Proc. EDO*, volume 1999 of *LNCS*. Springer, 2000.
  65. C. Liebig and S. Tai. Middleware mediated transactions. In *Proc. DOA'00*, 2001.
  66. J. Lopes de Oliveira, C. Bauzer Medeiros, and M. Cilia. Active customization of GIS user interfaces. In *Proc. IEEE Int'l Conference on Data Engineering (ICDE)*, pages 487–496, Birmingham, UK, April 1997. IEEE Computer Society Press.
  67. C. Ma and J. Bacon. COBEA: a CORBA-based event architecture. In *Proc. COOTS'98*, 1998.
  68. Microsoft Corp. BizTalk Framework 2.0: Document and Message Specification. Microsoft Technical Specification, December 2000.
  69. R. Motwani, J. Widom, and A. Arasu et al. Query processing, approximation, and resource management in a data stream management system. In *Proc. Int'l Conference on Innovative Data Systems Research (CIDR)*, 2003.
  70. G. Mühl. Generic constraints for content-based publish/subscribe systems. In *Proc. Int'l Conference on Cooperative Information Systems (CoopIS)*, volume 2172 of *LNCS*. Springer, 2001.
  71. G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, 2002.

72. G. Mühl, L. Fiege, and A.P. Buchmann. Filter similarities in content-based publish/subscribe systems. In *Proc. ARCS*, volume 2299 of *LNCS*. Springer, 2002.
73. G. Mühl, L. Fiege, F.C. Gärtner, and A.P. Buchmann. Evaluating advanced routing algorithms for content-based publish/subscribe systems. In *Proc. IEEE/ACM MASCOTS'02*, 2002.
74. Object Management Group. Event Service Specification. Technical Report formal/97-12-11, Object Management Group (OMG), Framingham, MA, May 1997.
75. Object Management Group. CORBA Notification Service Specification. Technical Report telecom/98-06-15, Object Management Group (OMG), Framingham, MA, May 1998.
76. Object Management Group. Management of event domains. Version 1.0, Formal Specification, 2001. formal/01-06-03.
77. Object Management Group. Enhanced View of Time Service, Version 1.1. Technical Report formal/02-05-07, (OMG), 2002.
78. K. O'Connell, T. Dinneen, S. Collins, B. Tangney, N. Harris, and V. Cahill. Techniques for handling scale and distribution in virtual worlds. In A. Herbert and A.S. Tanenbaum, editors, *Proc. 7th ACM SIGOPS European Workshop*, pages 17–24, Connemara, Ireland, September 1996.
79. B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The Information Bus – an architecture for extensible distributed systems. In *Proc. 14th Symposium on Operating Systems Principles (SIGOPS)*, pages 58–68, December 1993.
80. Object Management Group (OMG). Transaction service v1.1. Technical Report OMG Document formal/2000-06-28, OMG, Framingham, MA, May 2000.
81. L. Opyrchal, M. Astley, and J. Auerbach et al. Exploiting IP multicast in content-based publish-subscribe systems. In J. Sventek and G. Coulson, editors, *Proc. IFIP/ACM Int'l Conference on Distributed Systems Platforms and Open Distributed Processing(Middleware)*, volume 1795 of *LNCS*, pages 185–207. Springer, 2000.
82. Oracle, Inc. *Oracle Advanced Queuing (AQ)*, 2002.
83. Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proc. IEEE Int'l Conference on Data Engineering (ICDE)*, 1995.
84. N. Paton, editor. *Active Rules in Database Systems*. Springer, Berlin Heidelberg New York, 1999.
85. P. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In J. Bacon, L. Fiege, R. Guerraoui, A. Jacobsen, and G. Mühl, editors, *Proc. 1st Int'l Workshop on Distributed Event-Based Systems (DEBS'02)*, Vienna, Austria, July 2002. IEEE Press. Published as part of the ICDCS '02 Workshop Proceedings.
86. P. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *Proc. 1st Int'l Workshop on Distributed Event-Based Systems (DEBS'02)*, Vienna, Austria, July 2002.
87. S.P. Reiss. Connecting tools using message passing in the Field environment. *IEEE Software*, 7(4):57–66, July 1990.
88. RosettaNet. RosettaNet Implementation Framework: Core Specification v2.00.01. RosettaNet Technical Specification, March 2002.
89. A. Rowstron, A.M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In J. Crowcroft and M. Hofmann, editors, *Proc. 3rd Int'l Workshop on Networked Group Communication (NGC 2001)*, volume 2233 of *LNCS*, pages 30–43. Springer, 2001.
90. R. Schulte. A Real-Time Enterprise is Event-Driven. Research Note - T-18-2037. Gartner Group, 2002.

91. R. Schwarz and F. Mattern. Detecting casual relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 7(3), 1994.
92. S. Schwiderski. *Monitoring the Behaviour of Distributed Systems*. PhD thesis, Selwyn College, Computer Lab, University of Cambridge, 1996.
93. K.J. Sullivan and D. Notkin. Reconciling environment integration and software evolution. *ACM Transactions of Software Engineering and Methodology*, 1(3):229–269, July 1992.
94. S. Tai and I. Rouvellou. Strategies for integrating messaging and distributed object transactions. In *Proc. IFIP/ACM Int'l Conference on Distributed Systems Platforms and Open Distributed Processing(Middleware)*, New York, USA, April 2000. Springer.
95. W.W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A.P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In H.-Arno Jacobsen, editor, *In Online Proceedings of the 2nd Int'l Workshop on Distributed Event-Based Systems (DEBS'03)*, June 2003.
96. F. van Harmelen, J. Hendler, and I. Horrocks et al. OWL Web Ontology Language 1.0 Reference. W3C Working Draft, W3C, March 2003.
97. G. Weikum and G. Vossen. *Transactional Information Processing: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann, 2001.
98. M. West. The state of business rules. *MiddlewareSpectra*, 15(11), November 2001.
99. J. Widom and S. Ceri, editors. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.
100. S. Yang and S. Chakravarthy. Formal semantics of composite events for distributed environments. In *Proc. IEEE Int'l Conference on Data Engineering (ICDE)*, 1999.
101. D. Zimmer and R. Unland. On the semantics of complex events in active database management systems. In *Proc. IEEE Int'l Conference on Data Engineering (ICDE)*, 1999.

## **Part IV**

---

### **Personalized Access to the Web**



---

## Introduction

Access to the Web may be from a variety of devices and user interfaces, by different users at different locations and times, and for varying purposes. This chapter explores techniques for adapting Web information to the mode of access to it and to the user's specific requirements and preferences.

Chapter 15 begins with a survey of architectures for adaptive hypermedia (AH). Cannataro and Pugliese identify the three kinds of models that are relevant to AH systems: models of the application domain, models of users (also known as profiles), and models of adaption of domain models to user models. They identify three main aspects of adaptive functionality in AH systems – content adaption, link adaption, and presentation adaption. They describe some key existing AH architectures and systems, including a discussion of AH systems in mobile environments, before going on to describe in greater detail a specific XML-based AH architecture called XAHM.

XAHM encompasses all three forms of adaption: content, link, and presentation. Data sources and possible presentations are described using XML. The author of a hypermedia defines a logical graph structure for the hypermedia, assigning for each user profile a traversal probability to each arc of the hypermedia. A simulation tool is provided that can animate the hypermedia with respect to each user profile, allowing authors to modify the hypermedia or the profile. At run time, users' browsing behaviour is monitored in order to find the best match for each user from the set of profiles. One of the open problems discussed is development of AH techniques and systems which can adapt to applications, as opposed to just human users, accessing the hypermedia. Another important direction of further research is the synergy between adaptive hypermedia and Web services.

In their chapter, Cannataro and Pugliese identify three major applications for AH systems: on-line information systems, information retrieval, and educational hypermedia. The theme of educational hypermedia is continued in Chapter 16, which discusses adaptive Web-based educational hypermedia. De Bra et al. start by discussing how the advent of the Web is beginning to make possible learning that can take place 'anyplace, anytime, anyhow'. The Internet itself is providing the 'anyplace, anytime' aspects, while adaptive educational hypermedia (AEH) systems aim to provide the 'anyhow'. In particular, AEH systems provide adaption in three ways

for different learners: learning content (tailored to individual learners' current levels of knowledge), connectivity between fragments of content (allowing different navigation pathways through it for different learners) and learning culture (allowing for different styles of learning).

The authors discuss the technical requirements and architecture of AEH systems. They also identify and discuss the three kinds of models that are relevant to adaptive hypermedia – user models, domain models, and adaption models – but in this case specifically in the context of educational hypermedia. They describe the techniques used for these models in some major AEH systems. The authors then go on to describe a particular AEH architecture called AHA! in which event–condition–action rules are used to encode the adaption of user models, links, and content as users navigate through the hypermedia. They end by describing recent research into collaborative and incremental authoring of adaptive hypermedia, and into the use of ontologies to allow standardisation, validation, and sharing of knowledge about learning resources and users. The authors also note that the emerging metadata standards for learning objects (e.g., IEEE LOM, IMS, SCORM) are also likely to provide a way forward to better interoperability of AEH systems.

Chapter 17 picks up another of the themes from Chapter 15, namely AH systems in mobile environments, and examines techniques for personalising presentation in the face of the physical constraints of PDAs and mobile phones. Smyth and Cotter begin with a review of the technologies making up the mobile Internet. They observe that the hierarchical menu structure of typical mobile portals results in users having to spend a significant proportion of their time navigating through the menu structure (navigation time), as opposed to interacting with actual content (content time), which is their primary goal.

Smyth and Cotter present a predictive model for a user's navigation effort, the click-distance model, which is based on the number of 'navigation steps' needed to reach a specific item of content from a portal's home page. They argue that click-distance should be reduced by adapting the menu structure of a mobile portal to the needs of individual users, so that content of interest to a particular user is 'promoted' up the menu hierarchy. Their personalisation technique is based on a probabilistic model of user navigation preferences that can be used to predict the likelihood that some menu option  $o$  will be selected given knowledge of the user's prior selections and present position  $m$  in the menu hierarchy. The options  $o$  may be either directly or indirectly accessible from  $m$ , and when a user arrives at a menu  $m$  this probabilistic information is used to promote to  $m$  the user's  $k$  most likely next options from deeper within the menu hierarchy (the size of  $k$  being limited by the menu size). Moreover, within menu  $m$ , options are listed in descending order of their access probability.

The authors' empirical analysis of some live user interactions with mobile portals over a period of several weeks shows that this kind of personalisation can achieve a significant reduction in click-distance, coupled with a significant increase in content time. Open research problems include refining the user model to include additional parameters such as recency of access and time of day, and allowing multiple models of users since a user may access a mobile portal in different roles, e.g., for business and for leisure.

Chapters 15–17 discuss prediction of user behaviour and adaption of Web content in a specific hypermedia application or portal. Chapter 18 concludes this section, and the book, by exploring techniques for learning and predicting users' browsing behaviour in the Web as a whole. In particular, the chapter discusses techniques for building user models that will allow prediction of the  $n$ th Web page that a user will request given knowledge of his/her previous  $n - 1$  page requests, for some  $n$ . Such predictions can be used for optimising both Web server and Web browser performance by allowing prefetching and caching of Web pages, and also for making recommendations to users.

Davison begins with a review of the main issues and approaches to learning and predicting Web request patterns and describes several prediction algorithms that have been proposed. He discusses the three main types of workload model – proxy, client and server – to which prediction of Web requests can be applied, and describes six datasets containing Web request traces of each type. These datasets are then used to evaluate the predictive accuracy of a particular experimental prediction system that employs Markov-based probabilistic techniques. The empirical evaluation of this system examines the effect on predictive accuracy of increasing model parameters such as the number of predictions being made (i.e., the number of possible next actions), the value of  $n$ , and the size of the prediction window (i.e., how many user requests ahead are being predicted). A general conclusion is that larger models can outperform simpler ones.

Davison observes that changes in the Web content are likely to result in changes in users' Web request streams, and thus in building models of users' browsing behaviour emphasis should be put on recent browsing activity in order for the models to adapt to changes. One of the open research questions he identifies is the need for a more detailed investigation of the trade-off between the complexity and space cost of prediction models on the one hand and their predictive accuracy on the other.

---

# A Survey of Architectures for Adaptive Hypermedia

Mario Cannataro<sup>1</sup>, Andrea Pugliese<sup>2</sup>

<sup>1</sup> Università “Magna Græcia” di Catanzaro, Catanzaro, Italy  
cannataro@unicz.it

<sup>2</sup> DEIS – Università della Calabria, Rende, Italy  
andrea.pugliese@deis.unical.it

**Summary.** This chapter surveys and summarizes architectures and models for Adaptive Hypermedia (AH), discussing the main issues that arise and the commonly used techniques for adaption. An overview of some existing architectures and systems is given. We then focus in greater detail on one particular architecture, the XML Adaptive Hypermedia Model. We conclude with a summary and an attempt to foresee future directions in AH systems.

## 1 Introduction

*Adaptive Hypermedia* (AH) is a relatively young research area where the concepts of user modeling and (Web-based) hypermedia are combined to build systems capable to adapt themselves, in a proactive or reactive way, to different users’ characteristics. In fact, the personalization of presentations and contents, i.e. their adaptation to users’ requirements and goals, is becoming a major requirement in recent Web systems. The need for adaptation arises from different aspects of the interaction between users and hypermedia systems: (i) users to be dealt with are increasingly heterogeneous due to different interests and goals, world-wide deployment of information and services, etc.; (ii) hypermedia systems must be made accessible from different users’ terminals, which can differ not only at the software level (browsing and elaboration capabilities) but also in terms of ergonomic interfaces (scroll buttons, voice commands, etc.) and kinds of network; (iii) finally, as a result of the rapid evolution of the applications in this field, AH models and architectures must be able to support new technologies, media, devices, kinds of applications, and application-to-application protocols.

The goals of this chapter are to survey and summarize AH architectures and models, discussing their main issues, and to describe their main components and operations through the *XML Adaptive Hypermedia Model* (XAHM), an XML-based AH model and architecture. The rest of the chapter is organized as follows. Section 2 addresses the main requirements of AH systems, describes their main components and the basic information that such components exchange, and outlines the most commonly used techniques for adaptation. Section 3 gives an overview of some existing AH archi-

tructures and systems. Sections 4 and 5 describe XAHM and its software architecture. Finally, Section 6 summarizes the chapter and attempts to foresee future directions in AH systems.

## 2 Requirements and Architectures of Adaptive Hypermedia Systems

After presenting some basic terminology and concepts of AH systems, this section discusses their main components and operations. First, we introduce the models behind AH systems, then we discuss the techniques used to realize such models, and finally we present basic architectural components implementing AH systems.

### 2.1 What is an AH System and What is it Useful for?

An *adaptive hypermedia (AH)* system is defined as “an hypermedia system which reflects some features of the user in a user model and applies this model to adapt various visible aspects of the system” [13]. More particularly, AH systems can be distinguished into (i) *adaptable* hypermedia, in which the user provides his/her model, and contents are delivered with respect to the given model; (ii) *adaptive* hypermedia in the strict sense, where the system monitors the user’s behavior and adapts the presentation accordingly; and (iii) *dynamic* hypermedia, where there are no predefined presentations, so on the basis of the user’s model, the system generates presentations combining in a completely dynamic way atomic information fragments. The most common applications of adaptive hypermedia systems (AHSs) that have arisen so far can be summarized into three main categories [13]:

**Online information systems.** Along with classical information systems, these applications include online help systems, virtual museums, hand-held guides, e-commerce systems, electronic encyclopedias, and other kinds of applications that naturally require user-adapted interaction.

**Information retrieval.** In this field, the majority of the systems appearing recently have been designed to work on the Web. The most important challenge here is to avoid the restriction of the retrieval activity to a portion of the very large hyperspace considered.

**Educational hypermedia.** In this field, the introduction of the Web has greatly increased the number and type of systems developed. Recently, frameworks including authoring tools have appeared that are capable of building and supporting adaptive distance education courses.

### 2.2 Models Behind AH Systems

Three main high-level models are generally used to describe AH systems, with the objective of covering their main aspects and features:

- The *application domain model* (DM) is used to describe the hypermedia basic contents and concepts and their organization to depict more abstract concepts. This model also describes the different sources that affect the adaptation process, and must allow for an effective observation of users' actions, with respect to each particular application domain, in order to gather significant data for user modeling. The DM can allow authors to add metadata to the information fragments that is helpful for deciding which ones are useful for the user. The emerging Semantic Web [17] and the use of ontologies to classify Web documents is likely to give rise to new application domains for AH systems.
- The *user model* (UM) aims at describing users in terms of characteristics, preferences, expectations in the browsing of the hypermedia, and so on.
- The *adaptation model* (AM) is related to *content selection*, and *content* and *link adaptation*, i.e. it allows specification of how to manipulate information fragments and links presented to the user. The AM has to adapt the DM with respect to the UM. In other words, the adaptation process allows the DM to be viewed through the UM, built around a user by the selection and adaptation of useful contents.

To efficiently allow the realization of user-adaptable contents and presentation, a modular and scalable approach to describe and support the adaptation must be adopted. Application domain and adaptation models must allow the hypermedia to be described in such a way that it is easy to find all the variables that need to be supported in an adaptive way. User modeling must cope not only with the user's explicit behavior, e.g. browsing activity, but also with other implicit aspects regarding his/her environment and dynamic constraints. Finally, AH architectures must easily and efficiently support the adaptation process and must be flexible with respect to different kinds of adaptivity sources.

### 2.3 Information Sources for Adaptation

Information for building user models can be gathered by only observing users, thus adopting *automatic user modeling* (or *implicit acquisition*), or by allowing users to intervene in the process of modeling e.g. through content rating or questionnaires (*co-operative user modeling* or *explicit acquisition*). The information gathered is typically related to various aspects of users, which we summarize next.

**Personal characteristics.** Such characteristics comprise *knowledge* about the contents of the application domain, external *background*, *experience* within the system, *goals* (short-term objectives such as finding an information, learning a concept, obtaining a service, and so on), *preferences* (particular interest regarding some parts of the hypermedia), *interests* in general and longer-term, *individual traits* (personality factors such as introvert/extravert), *cognitive factors*, and *learning styles*. The user's behavior in this context corresponds to the actions he/she performs, such as the sequence of followed links (*clickstream*) and the set of performed transactions (*transaction stream*). It also corresponds to the knowledge shared with other users in collaborative systems, called *indirect social behavior*, or the interaction with do-

*main assistants* (or *agents*) that exploit their domain knowledge to support navigation, called *direct social behavior*.

**Environment and technology.** These aspects are related to the time–spatial location of the user and to the platform used, e.g. terminal and network. The importance of such aspects is increased by Web-based applications, particularly in the *ubiquitous computing* paradigm, demanding adaptation to the location the user resides at, to the time of the interaction, to platform, to network conditions, and so on, especially in order to satisfy response-time requirements.

The information gathered is used to build user models. Such models, also called *profiles*, are generally distinguished into two main categories. In *overlay models*, the user is described through a set of *attribute-value* pairs, where values are quantitative, e.g. percentage, or qualitative measures, e.g. poor/good/excellent. *Stereotype models* instead indicate that the user belongs to a group of users having common characteristics and are distinguished into *pure stereotypes*, indicating only the belonging to a group and not its characteristics, *mixed stereotypes*, which point out group characteristics through attributes, and *multiple stereotypes*, in which a user can be associated with different groups at the same time. The construction of user models employs concepts borrowed from many different fields. They are mainly derived from artificial intelligence (concept networks and nonsymbolic methods such as case-based reasoning, machine learning, and neural networks), from Data Mining (tree-based classification, mining of association rules, clustering), and from the field of statistics and probability (Bayesian classifiers).

Presented in [22] is another taxonomy for data used to build user models in which *personal* data (user's characteristics in a strict sense) is distinguished from *usage* data (choices made by the user not resolvable to personal characteristics) and *environment* data (comprising environment and technology of our taxonomy). The choice of one of these two taxonomies is unessential: they can be considered equivalent.

## 2.4 Adaptation Techniques

With regard to the most commonly used techniques to adapt presentations on the basis of domain and user models, three main areas exist:

**Content-level adaptation or adaptive content selection.** This area concerns the selection and composition of information fragments to be later presented to the user. Adaptation here regards *multimedia* (selection of different multimedia fragments), *text* (insertion, removal, alteration, and sorting of text fragments, use of *stretchtext*, i.e. small placeholders to be “opened” by users, text summarization, and natural language generation), and *modality* (choice among different *types* of media to present to the user, related to the same content, e.g. for a video we can have full video, still image, text description, or use of them in parallel).

**Link-level adaptation or adaptive navigation support.** This area concerns the manipulation of presented links. Many techniques are used for this purpose. *Direct guidance* presents to the user, at each node, a “next” link to the node which presumably is the most interesting at a given time. *Link sorting*, *hiding* and *annotation* essentially

change the visualization of a fixed set of links. Finally, *map adaptation* and *adaptive link generation* are the most complex kinds of link level adaptation. The former regards the complete re-organization of the overall link structure of the hypermedia. The latter includes the discovery of useful links between documents, the generation of similarity-based links between items, and the dynamic recommendation of relevant links.

**Adaptive Presentation.** This area is related to the management of different ways to present contents and links to the user. For instance, in this area decisions are taken about multimedia fragments having different resolutions and sizes, or about the annotation of text and links.

## 2.5 AH Architectures and Components

The architectures for the support of AH systems should support the following operations: design of the application domain, i.e. of AH contents; construction of the user model; application of adaptation techniques; and monitoring and tuning of the system, on the basis of usage observation. The software components implementing those operations are briefly described in the following.

**Authoring tools** (*what to adapt*). Authoring tools allow: (i) selecting the basic multimedia contents of the hypermedia; (ii) defining more abstract concepts as the basis of the “logical” hypermedia structure; (iii) specifying the basic information to be considered to build user models; and (iv) specifying the parts of the hypermedia on which to perform the adaptation.

**Profilers** (*adapt to what*). Profilers are components devoted to the observation of users and to their modeling, according to different strategies. Profilers comprise *observation modules* that collect data about users’ behaviors.

**Adaptation engines** (*how to adapt*). Adaptation engines deal with the generation of visible contents to be presented to users. Typically, their core part is a *transformation module* that composes basic information fragments starting from predefined templates and applying user models.

**Monitoring and self-tuning modules** (*how to evolve*). The level of adaptation typically increases as users’ heterogeneity increases, like other sources of adaptation, and as the system knows more about the history of its usage. In fact, in the design phase, all possible uses of the system could not be known. Therefore, often monitoring and self-tuning components are needed. Common techniques to take into account system usage use data mining [5].

Figure 1 shows the main logical flows in AH systems. In the figure, gray boxes show architectural parts that are peculiar to adaptive systems, while ovals and boxes represent software components and data, respectively.

## 3 An Overview of Adaptive Hypermedia Models and Systems

A number of interesting models and architectures have been developed in the last years for adaptive hypermedia systems. Their constant evolution is driven by increased



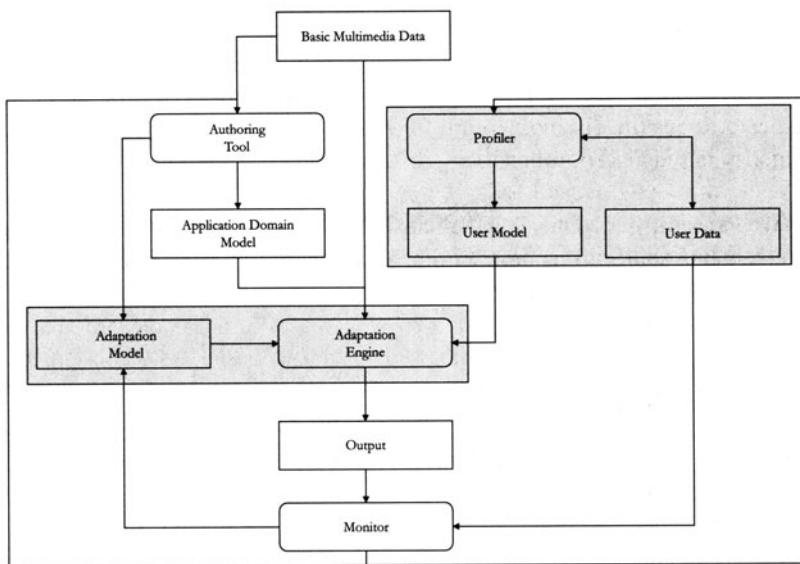


Fig. 1. Main logical flows in adaptive hypermedia systems

knowledge in the hypermedia field. In this section we briefly describe some models developed to specify and design AH systems, and we present an overview of systems in different application domains.

### 3.1 Models

Some of the most significant models developed in the past are essentially domain models, designed mainly for capturing the semantics found in hypermedia systems, while others deal with adaptivity in a more strict sense. Our belief is that the most promising approach to the modeling of application domains is the *data-centric* one, and many existing models already employ well-known data modeling techniques. Moreover, recent semistructured data models are particularly well-suited for representing information on the Web [1].

The *Dexter* hypertext model [20] was the most significant effort made in the early 1990s in order to capture typical abstractions in hypermedia and to provide a uniform terminology. The model is structured into three main layers. The *run-time* layer describes the mechanisms for supporting the interactions with users. The *storage* layer describes a “database” that is composed of a hierarchy of *components* connected by *links*; components contain data fragments, but are treated as generic containers in this layer. The *within-component* layer deals with the actual contents of components. Finally, in *Dexter* there are two “interface” layers, called the *presentation specifications* layer for encoding presentations into the storage layer, and the *anchoring* layer for addressing single items of components from the storage layer.

The *Hypermedia Design Model* (HDM) [19] and the *Object-Oriented Hypermedia Design Model* (OOHDM) [23] are intended essentially for capturing the semantics of

a hypermedia application domain. In HDM the main abstractions of an application are expressed by means of entities composed into components organized as trees. Navigation is internal to entities, among entities, or noncontextual (i.e. through indexes). With regard to the design process, HDM distinguishes between *in-the-large* design, regarding the overall structure and navigation aspects, and *in-the-small* design, dealing with layout and dynamics of the hypermedia. OOHDM is the object-oriented evolution of HDM. It introduces the classical aspects of this paradigm, e.g. abstraction and composition mechanisms. In OOHDM the navigational space is organized by means of appropriate abstractions, and navigational objects are views (in the database sense) over conceptual objects. The main design phases in OOHDM are: *conceptual design*, where a conceptual model of the application domain is built, also by separating navigational nodes from components supporting the computational aspects (such as algorithms and database access); *navigational design*, where a *navigational model* is defined as a view over the conceptual model; *abstract interface design*, for constructing *interface objects* that manage events generated by the interaction with users; and *implementation*, where conceptual, navigational, and interface objects are mapped into a particular run-time environment.

The *Intelligent Multimedia Presentations* reference model (IMMPS) [4] considers separately processes for handling interactions with users, and functional parts of the systems that maintain models, of domains, users, etc., called the *knowledge server*. Interactions with users are managed through five distinct layers, each of which interacts with the knowledge server. The *control layer* is used to resolve links, possibly starting from abstract concepts. The *content layer* selects the actual data fragment to be shown. The *design layer* composes a page from the selected fragments, while the *realization layer* finalizes the page so that it can be displayed by the browser, possibly applying an appropriate style sheet. Finally, the *presentation display layer* represents the rendering of the page, as performed by the browser.

*WebML* [16] is one of the most complete models for (adaptive) hypermedia. It defines a notation for the specification and the design of a hypermedia on the basis of some constructs, formally defined in XML and intended for describing the application context at a high level of abstraction. The design phase in WebML is carried out by composing a set of orthogonal abstraction models:

- *Structural model*, for the conceptual organization of data; WebML is compatible with classical notations such as the E/R and ODMG models, and UML class diagrams.
- *Hypertext model*, for the description of different hypertext views over the hypermedia (*site views*). The hypertext model consists of a *composition model*, for the organization in pages and each page in information units, and a *navigational model*, for the description of how relationships among data can be transformed into navigational options. The design phase related to the hypertext model is split into two distinct parts similar to the “in the large” and “in the small” ones of HDM.
- *Presentation model*, for the terminal-independent definition of the layout of pages.

- *Personalization model*, for the description of users and their organization in groups, and for the definition of *event-condition-action* rules for the hypermedia personalization.

The Dexter-based *Adaptive Hypermedia Application Model* (AHAM) [6] is based on a domain model describing the structure of the application, both at the conceptual level and at the level of data fragments and pages, by means of *atomic* and *composite nodes*. The overlay user model describes the user's knowledge of the concepts of the application domain. The *adaptation model* is composed of *adaptation rules* that indicate under which circumstances the user must be guided towards certain parts of the hypermedia. The *AHA!* system, entirely based on AHAM, has an architecture whose core part is the *adaptive engine*. It performs the construction of the presented pages and updates the knowledge values contained in the user model, on the basis of the user's browsing activity or test results.

Finally, our own XAHM model [14, 15] is fully described in Sect. 4. Further adaptation-oriented models, typically users' behavior-based and weakly data-centric, can be found in [7, 8, 9, 10, 11, 12, 21].

### 3.2 Architectures and Systems

Some significant AHSs of the recent past are summarized in Table 1. In what follows we describe some interesting recent developments regarding various application domains (see also Table 2).

**Adaptive hypermedia in wireless environments.** Mobile services present more competitive characteristics with respect to fixed-network services; among these are: mobile, time- and location-independent access to services (*anytime, anywhere*); different possibilities to reach the user (*asynchronous, synchronous*) and to obtain data (*push, pull*); and environment-aware services deployment (e.g. on the basis of the user's location, access time, etc.). However, to deploy applications in wireless environments it is necessary to take into account the constraints of wireless connections with respect to wired ones. The main constraints of wireless networks are low bandwidth, high latency, unsteady connection, high transmission *bit error rate* (BER), and unpredictable service availability. On the other hand, the main constraints of wireless terminals are small displays, limited user-input facilities, limited computational resources (CPU, RAM), and limited duration and power of batteries. Moreover, the availability of many models of terminals may cause interoperability problems.

The major requirements for adaptive personalization in wireless systems are quick learning of users' interests, and quick adaptation of contents to changing interests while avoiding brittleness caused by, for example, external conditions (e.g. a temporary disconnection). In a wireless environment the system should provide a good initial (nonpersonalized) experience and learn quickly the interests of new users. The benefits of adaptive personalization must emerge after the first few uses, and the transition from a nonpersonalized to a personalized experience must be smooth without random walks between presentations.

System Name	Application Domain	Adaptation Schema	User Model
ACE	Educational	Modality (CS)	Individual traits Preferences Abilities Learning style Context of work
Adaptive Hyperman	Information retrieval	Navigation limitation, guidance, link sorting (NS)	Background Current preferences (from feedback)
ADAPTS	Information systems Performance support	–	–
AHM	Educational	–	–
AIS	Information retrieval	Filtering (CS)	–
Anatom-Tutor	Educational	Text manipulation (CS)	Background Experience
ART-Web	Educational	–	–
Arthur	Educational	–	Individual traits (Non-symbolic modeling techniques)
AST	Educational	Modality (CS)	Preferences Abilities Learning style Context of work
AVANTI	Information systems Kiosks	Filtering, modality adaptation (CS)	Interests Preferences Abilities Learning style Context of work
Basar	Information systems Personalized views	Filtering (CS)	Goals Interests Background
C-Book	Educational	Conditional text and variants (CS)	Background
CAMELEON	Educational	–	–
CHEOPS	Educational	–	–
CID	Information retrieval	Navigation limitation, guidance, link sorting (NS)	Current preferences
DHS	Information retrieval	Guidance (NS)	Current preferences
ELFI	Information systems	Filtering (CS) Additional links (NS)	–
ELM-ART	Educational	Annotations (P)	–
HIPS	Information systems Handheld	Narration (CS)	Knowledge/Interests (from Location and/or movement)
Hy-SOM	Educational	Additional links (NS)	(Non-symbolic modeling technique)
Hypadapter	Educational Information systems	Text sorting and variants (CS) Annotations (P)	Background Absolute preferences
HYPERAUDIO	Information systems Handheld	Narration (CS)	Knowledge/interests (from location and/or movement)
HYPERCASE	Educational Information systems	Map adaptation (NS)	Goals
HYPERFLEX	Information retrieval	Link sorting, Guidance (NS)	Goals Current preferences
HyperTutor	Educational	Additional links, link hiding (NS)	–
HyPLAN	Information systems	Filtering (CS) Link hiding (NS)	Goals

System Name	Application Domain	Adaptation Schema	User Model
IfWeb	Information retrieval Browsing	Filtering, recommendations (CS) Annotations (P)	--
ILEX	Information systems Virtual Museum	Narration (CS)	--
ISIS-Tutor	Educational	Additional links, Guidance, Link hiding (NS) Annotations (P)	--
Letizia	Information retrieval Browsing	Filtering (CS)	--
Meta-Links	Educational	--	--
MetaDoc	Information systems	Stretchtext (CS)	Background Experience Knowledge
ORIMUHS	Information systems	Filtering (CS)	Goals
PEBA-II	Information systems Encyclopedia	Comparisons, Recommendations (CS) Additional links (NS)	Knowledge Interests (from browsing)
PowerBookmarks	Information retrieval Personalized Views	--	--
PUSH	Information systems	Stretchtext (CS) Link hiding (NS) Frame manipulation (P)	Goals Current preferences
RATH	Educational	--	--
Sitelf	Information retrieval Browsing	Filtering (CS)	--
Siteseer	Information retrieval Personalized views	Bookmarking (CS) Additional links (NS)	--
SKILL	Educational	--	--
SmartGuide	Information retrieval Search	--	--
SurfLen	Information retrieval	--	Collaborative
SWAN	Information systems	Filtering (CS)	--
Syskill and Webert	Information retrieval	Filtering (CS) Annotations (P)	--
TANGOW	Educational	--	--
TELLIM	Information systems E-Commerce	Modality adaptation (CS)	Platform Preferences Abilities Learning style Context of work
WebTagger	Information retrieval Personalized views	Bookmarking (NS)	--
WebWatcher	Information retrieval Browsing	Link sorting, Guidance (NS)	Current preferences

Table 1. AHSs of the recent past

AdaptiveInfo [2] has developed systems for adaptive access to Web systems whose contents are deployed to mobile users through wireless terminals. In particular, application domains such as classified advertising, news, entertainment directories, wireless portals, and voice portals are considered. The main challenge that must be faced is to fit the enormous amount of information that is available with the specific, unique needs of each user. Starting from a nonpersonalized view of the hypermedia, the system can quickly learn the user's interests and switch to a personalized view. An important aspect of content selection that emerges in many application domains is to

avoid content filtering, which could hide important and interesting contents because of imprecise user modeling and, in particular, because of changing interests.

**Adaptive hypermedia in information retrieval.** The *Dynamic Personalization Server* (DPS) [2] is an open, standard-based, and platform-independent tool produced by *humanIT*, which provides basic user modeling services to user-adaptive applications. The DPS has been used to personalize news services, creating a one-to-one relationship with users of the N24 German Web site. The main characteristic of DPS is that it can be seamlessly integrated into a preexisting publishing and content management architecture. To adapt a list of news to a user, a publishing system can query the DPS to obtain a user's profile. Using the retrieved user's profile, the relevant news items can be extracted by a content management system by matching the user's profile and news.

*CASPER* [2] is another example of AH developed for the information retrieval field, in particular for online recruitment services. In such a system the user has to search available positions by filling in a search form and querying the system. The main problems could be limited accuracy of the search results, when the query is incomplete because the user does not specify many details, or, on the other hand, strong reduction of the search results, when the user attempts to completely specify query fields. *CASPER* faces these problems by proposing a two-stage search engine that first selects job positions according to the query and then adapts query results with respect to user's interests. In summary, the main contributions of *CASPER* are improved retrieval accuracy based on user profile and a mix of client- and server-side operations.

*Broadcast News Navigator (BNN)* [2] is a question-and-answer system that can automatically segment, extract, and summarize news programmes, exploiting video, audio, and closed-caption text information coming from different sources. Question-and-answer systems allow users to perform searches by asking questions and getting personalized answers, in a way that differs from common search engines, where keyword-based searches typically get an overwhelming number of often irrelevant Web pages. Adaptive question-and-answer systems must also personalize answers to the user's interests. In BNN the user first poses a query and receives a *story skim* (a story skim contains, for each story matching the query, a key frame, the most frequent names appearing in it, and its source and date), then selects a story to get the details, e.g. its closed-caption text, named entities, a generated multimedia summary, or the full original video. The user's profile contains simple keywords or semantic entities such as individuals, locations, or organizations in which the user is interested. Moreover, the user can indicate media type preferences for device adaptation.

**Adaptive hypermedia in education.** *InterBook* and *AHA!* are two adaptive Web-based textbooks used in online education [2]. Usually, a learning application can be represented through (a hierarchy of) concepts and the user model in an overlay model, storing a knowledge value for each domain concept. When a user reads pages, the system updates the user's profile, assuming an increase of knowledge about the concepts associated with those pages, and decides whether a user is "ready" to study a new concept. *InterBook* uses two popular forms of adaptive navigation support: link

annotation and link hiding. Links marked with a red dot indicate pages not to be yet read by the user, green dots indicate recommended (ready) pages, while white dots indicate pages (possibly already read) containing concept(s) already known by the user. On the other hand, AHA! uses a link-hiding technique, where blue, purple, and black anchors respectively indicate ready and new pages, ready but not new pages, and not ready pages. It should be noted that the user can always reach all the links; thus AHA! avoids a problem common to some AHSs called *opaque behaviour*, that is, the user cannot reach potentially useful pages or contents because of incorrect adaptation. Another useful adaptation technique is the provision of additional explanation of concepts on the basis of the user model. Moreover, AHA! can support additional navigation adaptation techniques such as link sorting, link removal, and link disabling.

System name	References	Application domain	Adaptation schema	User model	Related fields
AdaptiveInfo	www.adaptiveinfo.com	Information retrieval	Content selection	Interests	Machine learning
Broadcast News Navigator	www.mitre.org	Information retrieval	Content selection, Presentation	Interests, Preferences	Natural language generation/ Processing
CASPER	www.ucd.ie	Information retrieval	Content selection	Interests	Machine learning
Dynamic Personalization Server	www.humanit.com	Information retrieval	Content selection	Interests	Machine learning
GUIDE	www.comp.lancs.ac.uk	Tourism	Content selection	Context, Interests, Preferences	Ubiquitous computing
Inhabited Market Place	www.dfki.de	E-commerce	–	Interests	Agents, Natural language generation
InterBook, AHA!	www.win.tue.nl	Education	Navigation support, Presentation	Knowledge	Intelligent tutoring systems
Medical Information Systems	www.cee.hw.ac.uk	Medicine	Presentation	Medical record	Natural language generation
SeTA	www.di.unito.it	E-commerce	Presentation	Interests, Knowledge	Agents, Natural language generation

Table 2. A summary of recent and emerging AHSs.

**Adaptive hypermedia in agent-based environments.** An emerging challenge in Adaptive Hypermedia is the personalization of character-based Web sites, i.e. systems where conversational characters such as virtual teachers, recommenders, agents helping users to fill in forms, etc., are used to present and suggest Web pages to the user. Whereas in conventional AH systems the adaptation regards content and presentation, in character-based ones the adaptation should also regard the agent’s presentation style and content. Moreover, the adaptive part of the user interface, which is conveyed by a character, is separated from the nonadaptive part, which is embedded into the Web

pages. This alleviates the opaque behavior problem of AH systems: each user sees the same Web-based environment, but nevertheless gets an individual tour helped by the adaptive agent.

The *DFKI (German Research Center for Artificial Intelligence) AiA (Adaptive InfoBahn Access)* system allows automating both the generation of customized Web pages and the creation of personalized scripts that are executed by the characters [2]. An evolution of the AiA system is the *Inhabited Market Place I*, a virtual place in which seller agents provide product information to potential buyer agents with the main goal to communicate facts about a certain product to an observing user. Whereas AiA presents each user with the same agent, the *Inhabited Market Place I* allows the user to form a team of agents with certain interests and personality features and to assign roles to them. Thus, the presentation conveyed by the characters is adapted on the basis of the character's personality, profile, and role assigned to the characters by the user. A further evolution of the system (*Inhabited Market Place II*) will allow a user to switch between a passive role, where he/she simply observes the characters, to an active one, where he/she can join a discussion. In summary, these three systems focus on various aspects of the adaptation process: the presentation style, the character's role and personality profile, and the amount of user engagement.

**Adaptive hypermedia in tourism.** The support of tourism applications is a classical field of AH systems. However, to meet the requirements of a particular kind of tourist, e.g. city visitors, the traditional adaptation process based only on a user model is not sufficient. The user's context, such as the user's current location and the history of visited places, should be taken into account into the adaptation process.

The *GUIDE* system [2] provides visitors to Lancaster in England with up-to-date and context-aware information. In *GUIDE*, hypermedia presented to visitors is adapted to both the environmental context (the major attractions in the city, their schedules, etc.) and the visitor's personal context (the visitor's current location, the visitor's interests, and the set of attractions already visited). Moreover, information is deployed through wireless terminals using a distributed network of servers, each serving a cell and thus providing updated information about the attractions around that cell. The adaptation process is based on a user model, which stores the user's personal preferences, and an environment model, which stores information about attractions within the city. As in other AH projects, the *GUIDE* system demonstrated that the vast majority of users do not want to employ too much time and effort in searching and retrieving information. Thus the use of context, such as the user's location, can help both to reduce this effort and to enhance the tourist experience. Moreover, it was confirmed again that it is better to use the environmental context for performing some kind of adaptation rather than use it to filter information.

**Adaptive hypermedia in e-commerce.** Another classical field of AH systems is the personalization in electronic commerce. Several *business to consumer* (B2C) applications employ one-to-one recommendation of products or are able to adapt presentations to users' device. *SeTA (Servizi Telematici Adattativi)* is a toolkit developed at University of Torino allowing the authoring of adaptive B2C applications [2]. The system is based on a multiagent architecture, and uses artificial intelligence techniques



for reasoning, i.e. Bayesian networks and rule-based systems. The adaptation process adapts content, layout, and presentation style to the customer's profile that comprises user's interests, knowledge about products, and user's behavior derived through the analysis of the clickstreams. The user model is initialized by using stereotypical information about customers. The configuration of the toolkit for different domains is supported by knowledge representation techniques.

**Adaptive hypermedia in medicine.** An emerging application domain for adaptive hypermedia is that of information systems tailored to users for which much information is already available and certified, for example medical information systems. In adaptive medical information systems, the patient's medical record can be used as a basis for the building of the user model, which can be built by combining the health history and condition of a patient with his/her browsing activity and current needs. At Heriot-Watt University, Edinburgh, an adaptive information retrieval system where information is selected, linked, and filtered on the basis of the patient's health record has been developed [2]. The system filters out irrelevant information and presents only the information related to the status of the patient, allowing the production of a synthetic and printable patient record.

## 4 XAHM: An Adaptive Hypermedia Model

This section presents our approach to the modeling of adaptive hypermedia. In the presented model, named *XML Adaptive Hypermedia Model* (XAHM), the logical structure and contents of an adaptive hypermedia are described along different logical levels. Upper (more abstract) layers are organized as graphs of concepts, whereas lower (more physical) layers are composed of XML documents describing individual pages of the hypermedia. Such pages include basic multimedia fragments extracted from different data sources and described by metadata. Moreover, XAHM uses a probabilistic approach for characterizing some aspects of the logical structure of the hypermedia, and a user classification algorithm for associating users with stereotype profiles. Finally, the adaptation model is based on a multidimensional *adaptation space*.

XAHM is specifically concerned with a complete and flexible data-centric support of adaptation. It is focused on (i) the description of structure and contents of an adaptive hypermedia in such a way that it is possible to easily point out the components on which to perform adaptation; (ii) a characterization of the hyperlinks useful to single out users' preferences and/or goals in a noninvasive way; (iii) a simple representation of the logic of the adaptation process, distinguishing between adaptation driven by technological constraints and adaptation driven by users' needs; and (iv) the support of a wide range of adaptation sources.

The lifecycle of XAHM-based hypermedia systems resembles the general one presented in [18], and comprises six macro activities:

- The *requirements analysis* is the identification of prospective users and the definition of the nature of the information base. In particular, the expected categories of users supposed to interact with the system are identified, and the technological and environmental conditions to be considered are singled out.
- The *conceptualization* of the application is done through a set of abstract models and metadata representations of data sources. Since we are dealing with Web applications, the focus is on capturing objects and relationships as they will appear to users.
- *Prototyping and verification* are essentially concerned with the validation of the probabilistic link structure of the hypermedia, and with the identification of further parameters on which to apply the adaptation.
- The *design* comprises the mapping of conceptual schemas into lower-level representations, to be stored in a repository. The three fundamental design dimensions are: the *structure*, describing the organization of the information and the semantic relationships (comprising the *logical structure design* and the *data design*); the *navigation*, capturing ways to access data and to move across contexts; and the *presentation*, expressing the way in which contents and navigation facilities are presented to the user.
- The *implementation* consists of simply filling in the modeled data sources with actual contents. This is different from the approach presented in [18], though, in XAHM final pages are constructed at run time only.
- *Evolution and maintenance* aim at finding appropriate changes to be applied at higher levels in the development chain and propagated to lower levels.

We chose to adopt XML primarily because of its flexibility and data-centric orientation. These characteristics it make it possible to elegantly describe data access and dynamic data composition functions, allowing the use of preexisting multimedia basic data, e.g. stored in relational databases and/or file systems, and the description of contents in a platform-independent way.

In the remainder of this section we first introduce the proposed multidimensional adaptation model. Then, we detail our layered domain model and the use of XML for expressing metadescriptions about basic information fragments and “neutral” pages to be transformed and delivered. Afterwards, we describe the proposed probabilistic interpretation of the hypermedia link structure and briefly outline the modeling phases of XAHM-based systems. Finally, we detail our user classification algorithm.

#### 4.1 The Adaptation Space

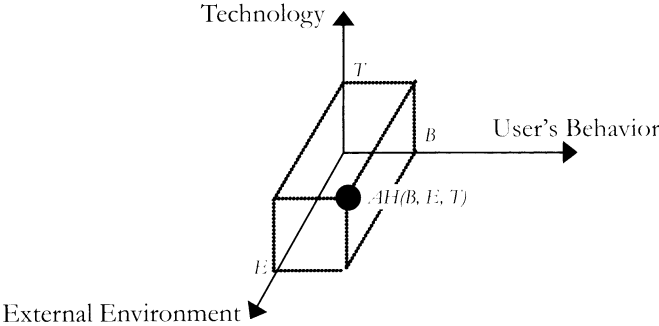
The application domain is modeled in XAHM along three abstract orthogonal adaptivity dimensions (Fig. 2):

- *user's behavior* (browsing activity, preferences etc.)
- *external environment* (time–spatial location, language, sociopolitical issues, status of external Web sites, etc.)
- *technology* (kind of network, bandwidth, quality of service, user's terminal, etc)

The position of the user in the adaptation space is denoted by a tuple of the form  $[B, E, T]$ . The values of  $B$ ,  $E$ , and  $T$  vary over a finite alphabet of symbols. The  $B$  value, related to the user’s behavior dimension, captures the stereotype profile the user is associated with. The  $E$  and  $T$  values identify environment location and used technologies, respectively. As an example,  $B$  could vary over  $\{novice, expert\}$ ,  $E$  over  $\{springSummer, autumnWinter\}$  and  $T$  over  $\{HTMLlow, HTMLhigh, WML\}$ .

The adaptive hypermedia is a subspace of the overall adaptation space, specified by the author by providing its description for some particular portions of the space. In turn, personalized views over the application domain are associated with different subspaces of the hypermedia.

The AHS monitors the different possible sources that can affect the position of the user in the adaptation space and collects a set of values, called *user*, *technological* and *external variables*. The decision of what variables to consider, made by the author of the hypermedia, depends mainly on the specific application domain.



**Fig. 2.** Adaptation space and adaptivity dimensions

On the basis of the above variables, the system identifies the position of the user. In particular, the parameter regarding the user stereotype profile is evaluated by an algorithm that makes use of a probabilistic interpretation of the link structure of the hypermedia; the algorithm is shown in Sect. 4.7. Note that we will actually consider many *dimension parameters* for denoting the position of the user. The distinction over three dimensions is used for grouping similar parameters, for abstraction purposes. A three-dimensional position of the user, however, can always be obtained by means of a simple mapping from the corresponding parameters to monodimensional values.

The adaptation model comprises both content and link adaptation; these are obtained by instantiating “neutral” pages, i.e. applying to them some constraints bound to the user’s position in the adaptation space. The user’s behavior and external environment dimensions mainly drive the generation of page *contents* and *links*, while the technology dimension mainly drives the adaptation of page *layout* (presentation layout, such as shapes of presentations or buttons; size of transmitted data, such as size

of extracted text, image and video resolution; kind of transmitted data, such as synthesized speech versus plain text, or uncompressed versus lossy compressed data). For example, an e-commerce Web site could show a class of products that fits the user's needs, as deduced from his/her behavior, applying a time-dependent price, e.g. night or day, formatting data with respect to the user's terminal, and sizing data with respect to network bandwidth. In the above example, a personalized view over the application domain might correspond to the point [*expert*, *autumnWinter*, *HTMLhigh*].

The application domain model remains abstract with respect to the alphabets of labels composing the domains of the dimension parameters. This feature is significant for the extensibility of the model, i.e. when an author needs to make the dimension variables feasible for a particular domain. In the above example, with respect to the technology dimension, the author could freely split the *WML* plane into two distinct planes *WMLhigh* and *WMLlow*. Such extensions are reflected into new views over the application domain, obtained considering the new parameters and modeling presentations subsequently.

## 4.2 Modeling the Logical Structure of a Hypermedia

The proposed application domain model uses a layered data model for describing the logical structure of the hypermedia. We use the notion of general *directed* graph (*digraph*), e.g. a graph in which more than one directed edge can connect two nodes, to capture the organization of some layers of the model.

The domain model has the following structure. At the lowest level there are *information fragments* (IFs) or *atomic concepts*, like texts, sounds, images, videos, etc. The information fragments can be stored into local and remote databases and/or file systems, and in general everywhere in the Web. In fact, very often these are pre-existing data and it would not be suitable to convert their formats. Furthermore, data can be structured, semistructured or unstructured as it is provided by different kinds of sources, such as object-relational databases, XML and HTML documents, texts, files, etc. The IFs are described by XML metadata whose structure is shown in Sect. 4.3.

*Presentation descriptions* (PDs) are “neutral” pages that capture the so-called *page concepts* [24] and correspond to the nodes of the hypermedia. They comprise multimedia contents, presentation layout, etc. Each component of the pages is associated with a portion of the adaptation space. Included information fragments are referenced by means of the XML metadata describing them. Formally, a PD is a tuple of the form

$$PD = \langle name, reference, components \rangle .$$

The *reference* is a numerical value used to create links from other PDs. *Components* is an ordered set of tuples of the form  $\langle content, dimensionParameters, linkDestination \rangle$ . The *content* can be plain text, some embedded code in a target language, e.g. HTML, or the reference to an information fragment. The *dimensionParameters* are used to associate the component with a portion of the adaptation space. The *linkDestination* is used to associate an hyperlink, either external or internal (towards other PDs), with the component. A PD can also *extend* other ones, inheriting their

components. Presentation descriptions are represented using XML; the structure of such documents is shown in Sect. 4.4. The final pages composed of actual fragments, also called *presentation units*, are dynamically generated at run time in a target language such as XML, HTML, WML, VoiceXML, synthesized speech, etc.

*Elementary abstract concepts* (EACs) represent larger units of information. They are sets of presentation descriptions organized in a digraph. Arcs represent simple relationships between page concepts and/or navigation requirements, e.g. a sequence of simple concepts to be learned before learning more complex ones. Arcs are differentiated with respect to users' stereotype profiles and are annotated by a weight used to represent their relevance with respect to each other. Formally, an elementary abstract concept is a tuple of the form

$$EAC = \langle name, reference, pds, arcs, entryPoints, exitPoints \rangle$$

where *pds* is the set of PDs composing it, and *arcs* is a set of tuples of the form  $\langle i, j, k, w \rangle$ , where *i* is the starting PD, *j* is the destination PD, *k* is the profile the arc is associated with, and *w* is a weight. *EntryPoints* and *exitPoints* are used for inter-EAC connections. An EAC can also extend other ones, inheriting all but the names.

*Abstract concepts* (ACs) are tuples of the form

$$AC = \langle C, arcs \rangle$$

where *C* is a set of EACs (or ACs themselves), and *arcs* is a set of tuples of the form  $\langle i, j, k \rangle$ , where *i* is the starting EAC exit point, *j* is the destination EAC entry point, and *k* is the profile the arc is associated with. Arcs represent relationships between concepts. They are differentiated with respect to stereotype profiles, but no weight is associated with them as the user is not supposed to perform any choice on them. The overall *domain model* (*DM*) is itself an abstract concept.

The (elementary) abstract concepts are differentiated with respect to user's profile only, because they need to be directly described by the author and it would not be suitable to build and maintain a large variety of weighted digraphs. However, future extensions of the system could support the personalization of the link structure of the hypermedia with respect to technological variables, so allowing "lighter" (with shorter paths) versions of the hypermedia to be browsed in a more agile way.

### 4.3 Metadescriptions of Information Fragments

In XAHM, each data source is wrapped by an XML metadescription. The use of metadata is a key aspect for the support of multidimensional adaptation. For example, an image could be represented using different levels of detail, formats, or points of view, whereas a text could be organized as a hierarchy of fragments, represented using different languages. Each representation of data could be associated with a portion of the multidimensional adaptation space. Furthermore, by means of metadescriptions,

data fragments of the same kind can be treated in an integrated way, regardless of their actual sources. In the construction of pages the author refers to metadata, thus avoiding access to fragments that is too low-level.

A number of XML Schema definitions [17] for the XML metadescriptions have been designed. They comprise descriptions of text (organized hierarchically), object-relational database tables, queries versus object-relational data, queries versus XML data (expressed in XQuery [17]), video sequences, images, XML documents, and HTML documents. The use of “pure” XML instead of more widespread formalisms for metadata, such as the RDF [17], is due to the fact that XML allows a simpler and more direct support to the proposed multidimensional approach, and that RDF is appropriate essentially for Web-available metadata.

As an example, consider the following metadescription of a query versus a relational database:

```
<query alias="imagequery" IP-address="..." port="..."
  database-name="..." username="..." password="...">
  <SQL-statement>
    select jpeg from image where key=#key
  </SQL-statement>
  <column name="jpeg" type="..." />
</query>
```

In such a metadescription, besides attributes regarding the connection to the database management system (DBMS), the key elements are the SQL statement (possibly with some parameters, #key in the example) and a description of the columns of the resulting object-relational table. Here, the only column contains objects of a complex type representing a JPEG image. Specific image fragments extracted by means of the above-shown query could be described as follows:

```
<image alias="image 1">
  <instance alias="high-res" description="hres version"
    location-type="query"
    location="imagequery(key=15).jpeg"
    mime-type="image/jpeg" />
  <instance alias="low-res" description="lres version"
    location-type="query"
    location="imagequery(key=17).jpeg"
    mime-type="image/jpeg" />
</image>
```

It should be noted that the metadescription of a JPEG image is abstract with respect to its actual source, referred to by means of location attributes. Furthermore, many instances of the same image can be differentiated within the same metadescription (in the example above, instances having different resolutions). “Complex” metadescriptions have been designed to allow direct referencing of single information fragments

by means of aliases and dot-notation; in the previous example, the two instances could be referenced simply as “image 1.high-res” and “image 1.low-res”.

#### 4.4 Presentation Descriptions

Each PD is a sequence of components, so its XML representation is a sequence of XML elements. The key elements of the PDs are the *plain-text* element, which is used to include text into pages; the *fragment* element, for including basic information fragments referenced by their aliases; and the *embedded-code* element, which allows inserting terminal-dependent code into the page (obviously, wrapped by an XML CDATA section).

As an example, consider the following XML description referring to the example of metadata presented in Sect. 4.3:

```
<presentation-description name="Picture"
  reference="13">
  <fragment reference = "image 1.high-res"
    networks="high" terminals="HTML"/>
  <fragment reference = "image 1.low-res"
    networks="low" terminals="HTML"/>
  <plain-text terminals="WML">
    This is a placeholder
  </plain-text>
  <embedded-code terminals="HTML">
    <[CDATA[<center>]]>
  </embedded-code>
  <plain-text profiles="expert" terminals="HTML">
    Long text 1
  </plain-text>
  <plain-text profiles="novice" terminals="HTML">
    Long text 2
  </plain-text>
  <embedded-code terminals="HTML">
    <[CDATA[</center>]]>
  </embedded-code>
  <plain-text terminals="WML">Short text</plain-text>
</presentation-description>
```

In this description, the presented picture is differentiated with respect to two parameters: *networks* (a high-resolution picture for a high-bandwidth network) and *terminals* (a placeholder for a WML terminal). The comment associated with the picture is differentiated with respect to profiles and terminals.

The dimension parameters can be any XML NMTOKEN, and the author is allowed to define the alphabet of labels regarding such parameters. As said before, this means that the model does not set up in advance the domains of the adaptivity dimensions, and it is always possible to extend them for a specific application domain.

#### 4.5 Probabilistic Interpretation of the Hypermedia Structure

In the layered model presented in Sect. 4.2, we introduced a weight in the graphs representing elementary abstract concepts to express links' relevance with respect to each other. In this section, we propose a probabilistic interpretation of arcs' weight, which is also used for the characterization of "latent" properties of user's behavior.

We consider the layered logical structure of the domain model to be mapped into the "plain" weighted digraph of presentation descriptions. So, a domain model with a set  $M$  of profiles is considered as a set  $N$  of presentation descriptions, where the generic description  $i \in N$  contains, for each profile  $k \in M$ , a set  $L_{ik}$  of annotated outgoing links  $(i, j, k)$ , where  $j$  is the destination node. The domain model is mapped into a weighted digraph  $G$ , where each node corresponds to a description and each directed arc to an outgoing link:

$$G = (N, E), \text{ with } E = \bigcup_{i \in N, k \in M} L_{ik}$$

The digraph  $G$  is also viewed as a set of weighted graphs  $G_k, k \in M$ , obtained by extracting from  $G$  nodes and arcs associated with each profile. Each  $G_k$  is named a *logical navigation graph*:

$$G_k = (N_k, E_k), \text{ with } N_k = \{i | (i, j, k) \in E \vee (j, i, k) \in E\} \\ \text{and } E_k = \{(i, j) | (i, j, k) \in E\}$$

Our probabilistic interpretation assumes that the weight  $W_k(i, j)$  of the arc  $(i, j)$  in  $E_k$  is the conditional probability  $P(j|k, i)$ , namely the probability that a user associated to profile  $k$  follows the link to node  $j$  having already reached node  $i$ :

$$W_k(i, j) = P(j|k, i), \text{ with } (i, j) \in E_k, k \in M$$

$P(i|k, i)$  is considered to be always zero. Obviously, for each node the sum of the weights of outgoing arcs for each profile must be always one.

We define a path  $S$  in  $G$  as the ordered set of arcs

$$S = \{(S_j, S_{j+1}, profile_j) | (S_j, S_{j+1}, profile_j) \in E, j = 0, \dots, l-1\}$$

where  $profile_j \in M$  represents the profile the user is associated with when he/she reaches node  $S_j$ . Note that paths involving different logical navigation graphs are allowed, since a user can be moved within different profiles during his/her browsing activity. The probability that a user associated to profile  $k$  follows the  $S$  path is

$$P_S^k = \prod_{j=0, \dots, l-1} W_k(S_j, S_{j+1})$$



That is,  $P_S^k$  is the product of the probabilities associated with the arcs composing the  $S$  path. The “shortest” path  $\tilde{S}_{ij}^k$  between two nodes  $i$  and  $j$  for a given profile  $k$  is the path with the maximum joint probability defined as

$$\tilde{P}_{ij}^k = \max_{S_{ij}^k} (P_{S_{ij}^k}^k)$$

where  $S_{ij}^k$  is the generic path between the nodes  $i$  and  $j$  through arcs associated with the profile  $k$ . Note that the “shortest” path between each pair of nodes, and for each profile, can be computed once and for all.

#### 4.6 Modeling Phases

In this section we detail the different phases through which an XAHM-based adaptive hypermedia is constructed. Three main phases can be identified, related to the layered model of Sect. 4.2.

**Semi-automatic metadata creation.** Since basic multimedia information fragments are always accessed by means of associated metadata, the first phase is concerned with the creation of such metadata. Our proposed architecture, which we describe in detail in Sect. 5, features a semiautomatic way to describe data sources, by means of an automatic exploration integrated by knowledge provided by the author. As an example, some components of the architecture are able to connect to local or remote DBMS and automatically extract the structure of relational or object-oriented tables. Or, they can explore local or remote file systems and extract metadata about stored files of known types. The author of the hypermedia is allowed to integrate such metadata, e.g. with human-readable explanations, or to create new ones, e.g. descriptions of typical queries.

**High-level structure definition.** The high-level structure of an adaptive hypermedia is modeled by means of the upper two layers of the graph-based model described in Sect. 4.2. After having defined the set of stereotype users’ profiles, the author defines abstract concepts, and finally describes EACs, specifying sets of PDs, adding to arcs the appropriate probabilistic weights. Obviously, the hypermedia designer could instead choose a bottom-up approach, starting from the definition of the EACs. Furthermore, the author is allowed to reuse (parts of) previously designed hypermedia, e.g. for different application domains.

**Presentation descriptions construction.** The last (and typically longest) phase of the AH design is the construction of the presentation descriptions. Here, the author composes basic information fragments, referencing their metadata, and associates them with specific portions of the adaptation space by means of dimension parameters. Thus, the author defines the subspaces of the adaptation space to which correspond views over the application domain.

#### 4.7 User Classification

The probabilistic interpretation of the link structure is used to characterize “intrinsic” properties of the hypermedia structure and “implicit” properties of the user’s behavior.

Such properties, related to the user's behavior adaptivity dimension, are expressed by means of an association of the user with a stereotype profile. In this section we describe our approach to such classification tasks.

The proposed algorithm builds a discrete *probability density function* (PDF)  $A(k)$ , with  $k \in M$ , measuring the “belonging probability” of the user to each group, i.e. how much each profile fits him/her. While the user browses, the system updates  $A(k)$  and the user's profile is consequently changed. In other words, on the basis of the user's behavior, the system dynamically attempts to assign the user to the best-fitting profile.

First, the algorithm considers some *static* (or *intrinsic*) *properties* of the probabilistic link structure expressed, for each profile  $k$ , by the following values:

- the mean value of the probability of the “shortest” paths in  $G_k$ ; high values of this term indicate the existence of highly natural paths in the hypermedia
- the mean value of the length of the “shortest” paths in  $G_k$ ; high values of this term mean longer natural paths in the hypermedia, which could be an advantage in the overall personalization process
- the number of nodes associated with the profile  $k$

Formally, these values are taken into account, constructing three corresponding PDFs:

$$\mu(k) = \frac{\sum_{q=1}^{|M|} \frac{\sum_{(i,j) \in E'_q} \tilde{P}_{ij}^q}{|E'_q|} \delta(k-q)}{\sum_{q=1}^{|M|} \frac{\sum_{(i,j) \in E'_q} |\tilde{S}_{ij}^q|}{|E'_q|}}, \quad n(k) = \frac{\sum_{q=1}^{|M|} |N_q| \delta(k-q)}{\sum_{q=1}^{|M|} |N_q|},$$

$$p(k) = \frac{\sum_{q=1}^{|M|} \frac{\sum_{(i,j) \in E'_q} |\tilde{S}_{ij}^q|}{|E'_q|} \delta(k-q)}{\sum_{q=1}^{|M|} \frac{\sum_{(i,j) \in E'_q} \tilde{S}_{ij}^q}{|E'_q|}}$$

where  $E'_k$  is the set of arcs in the transitive closure of  $G_k$ . Then a weighted mean of such functions, expressing the “intrinsic relevance” of profiles, is computed:

$$s(k) = \frac{\beta_0 \mu(k) + \beta_1 n(k) + \beta_2 p(k)}{\beta_0 + \beta_1 + \beta_2}$$

Here,  $\beta_i$  are suitable weights that measure the relative importance of the terms in  $s(k)$ . It should be noted that  $s(k)$  could change over time: the hypermedia structure can dynamically be updated, by adding or removing nodes or arcs, or changing their weights, on the basis of a semiautomatic observation of the behavior of many users or on the basis of an increased knowledge of the application domain by the author.

The dynamic properties of the user's behavior are captured by tracing his/her browsing activity. Browsing starts from the presentation unit associated with a starting node. If the user is already registered, the last  $A(k)$  is set as current. Otherwise, a generic profile is assigned to the user, or one is calculated on the basis of a questionnaire (see e.g. [3] for an interesting proposal for a probabilistic evaluation of a questionnaire). The initial value of  $A(k)$  is called  $A_0(k)$ . When the user visiting a node follows a link, the system computes the new PDF  $A'(k)$  on the basis of the collected user behavior variables and of  $s(k)$ , and then it decides the (new) profile to be associated with the user. To avoid continuous profile changing, it is possible to keep a profile for a given duration (i.e. number of traversed links), evaluating the  $A'(k)$  distribution at fixed intervals only.

The classification algorithm considers the path  $R = \{R_1, \dots, R_{r-1}, R_r\}$ , composed of the  $r$  most recently visited nodes, where  $R_{r-1}$  is the current node and  $R_r$  is the next node chosen by the user, and the visiting times on such nodes,  $T(R_1), \dots, T(R_r)$ . Then it evaluates, for each profile  $k$ :

- $P_R^k$ , the probability of having followed the  $R$  path through arcs associated with the profile  $k$ . A high value of  $P_R^k$  indicates that the visited nodes in  $R$  are relevant for profile  $k$  as the actual path is "natural" for the profile  $k$ .
- $\tilde{P}_{R_1, R_r}^k$ , the "reachability" of the next node  $R_r$  starting from the first node  $R_1$ , through arcs associated with profile  $k$ . This value takes into account the way the user *could have reached* the next node: a high "reachability" of  $R_r$  for profile  $k$  means that the user would have reached the next node in a "natural" way by following links associated with the profile  $k$ .
- $D^t[k]$ , the sum of the time units spent on the nodes from  $R_1$  to  $R_{r-1}$  associated with the profile  $k$ . As  $D^t[k]$  shows how the time spent on pages is distributed with respect to profiles, we consider this an indicator of the *interest* the user has shown. Significant visiting times must be accurately computed, so they are "vertically" normalized. Time units are in fact divided by a *score* that is proportional to the actual bandwidth (when applicable, i.e. when it can be probed) or that is equal to the sum of the time units spent on a certain number of recent nodes. A "horizontal" normalization, referring to the time spent on the same nodes by other users, appears to be unfeasible due to the extreme variability of presented pages [15].

Only the most recently followed  $r - 1$  links ( $r$  nodes) are considered, basically to avoid an "infinite memory" effect. In fact, considering  $R$  since the initial node, the probability  $P_R^k$  of having followed  $R$  in the profile  $k$  would be zero if the user visited just one node not associated with the profile  $k$ , as obviously  $W_k(i, j) = 0$  if  $(i, j) \notin E_k$ .

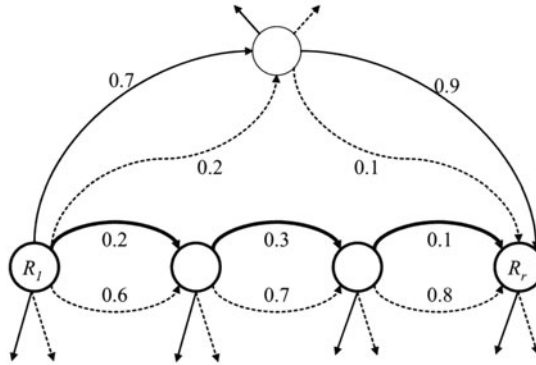
Formally, the computed values are taken into account to construct three corresponding PDFs:

$$c(k) = \frac{\sum_{i=1}^{|M|} P_R^i \delta(k-i)}{\sum_{i=1}^{|M|} P_R^i}, r(k) = \frac{\sum_{i=1}^{|M|} \tilde{P}_{R_1, R_r}^i \delta(k-i)}{\sum_{i=1}^{|M|} \tilde{P}_{R_1, R_r}}, t(k) = \frac{\sum_{i=1}^{|M|} D^t[i] \delta(k-i)}{\sum_{i=1}^{|M|} D^t[i]}$$

Finally, a weighted mean expressing the “dynamic relevance” of the profiles is computed:

$$d(k) = \frac{\alpha_0 c(k) + \alpha_1 r(k) + \alpha_2 t(k)}{\alpha_0 + \alpha_1 + \alpha_2}$$

Here,  $\alpha_i$  are suitable weights that measure the relative importance of the terms in  $d(k)$ . Note that it should be considered the possibility of trading off the effects of  $c(k)$  and  $r(k)$  on  $d(k)$ . In fact, the former takes into account the actual path so it aims to move toward profiles corresponding to local preferences, whereas the latter aims to disregard local choices, as the “shortest” paths do not necessarily involve the visited nodes between  $R_1$  and  $R_r$ . For instance, consider the situation depicted in Fig. 3. The actual followed path (in bold) is likely for the profile indicated with the dashed line, but there also exists an alternative path in the other profile, though not followed by the user, which connects  $R_1$  and  $R_r$  in a very “natural” way.



**Fig. 3.** An example of followed path

The algorithm for the evaluation of the new belonging probabilities takes as input the discrete PDFs  $A(k)$ ,  $A_0(k)$ , and  $s(k)$ ; the recently followed path, composed of the  $r$  most recently visited nodes; and the time spent on the  $r$  most recently visited nodes. Its output is a new probability density function  $A'(k)$ . The new PDF is evaluated by first computing the new discrete PDF  $d(k)$ , then applying the following formula:

$$A'(k) = \frac{\gamma_0 A_0(k) + \gamma_1 A(k) + \gamma_2 d(k) + \Delta \gamma_3 s(k)}{\gamma_0 + \gamma_1 + \gamma_2},$$

where  $\Delta = 1$  if  $s(k)$  has changed and  $\Delta = 0$  otherwise. Therefore, the new  $A'(k)$  is computed as a weighted mean of four terms. The first term expresses user's initial choices, the second term considers the story of the interaction, the third term captures the dynamic of the single user, and the fourth term expresses structural properties of the hypermedia, mainly depending on its topology. An high value of each of the terms in  $A'(k)$  expresses a high relevance of the profile  $k$ , so  $\gamma_i > 0$ . The new profile is chosen referring to the highest  $A'(k)$  value, or making a random extraction over the  $A'(k)$  distribution, on the basis of the author's choices.

## 5 An Architecture for the Support of XAHM-Based Hypermedia Systems

In this section we present an architecture for the run-time support and the authoring of XAHM-based systems.

### 5.1 The Run-Time System

The run-time system supporting XAHM has a three-tier architecture (Fig. 4), comprising the *user*, *application* and *data* layers. The user layer receives the final pages to be presented and scripts or applets to be executed, which are useful, for example, for detecting local time, location, available bandwidth, or for evaluating the time spent on pages. The user's terminal and the terminal software (operating system, browser, etc.) are typically communicated by the terminal *user agent*. At the application layer there are two main modules: the *adaptive hypermedia application server* (AHAS) and the *user modeling component* (UMC); they run together with a Web Server. The UMC maintains the most recent actions of the user and executes the algorithm for the evaluation of the user's profile.

The AHAS executes the following steps:

1. communicates to the UMC the most recent choice of the user
2. extracts from the XML repository the PD to be instantiated
3. receives from the UMC the new profile the user must be associated with
4. extracts basic data fragments from the data sources and composes them, on the basis of the known user's position
5. extracts the terminal-dependent XSL stylesheet from the XML repository
6. applies the stylesheet to the XML resulting page
7. returns the final page to the Web Server

The data layer stores persistent data and offers efficient access primitives. It comprises the *data sources level*, the *repository level* and a *data access module*. The data

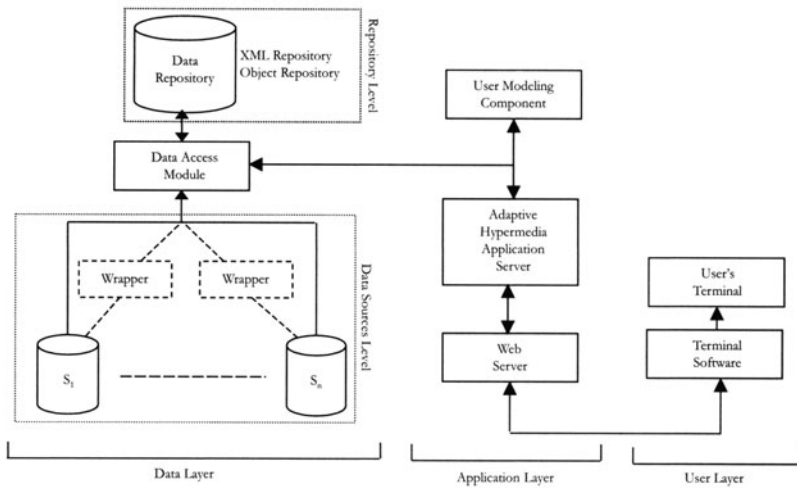


Fig. 4. The run-time system

sources level is an abstraction of the different kinds of data sources used to build the hypermedia. Each data source  $S_i$  is also accessed by a wrapper component, which generates in a semiautomatic way the XML metadata describing the data fragments stored in  $S_i$ . The repository level stores data provided by the data source level or produced by the author, comprising: XML documents into an *XML repository* (presentation descriptions, metadata, and XSL stylesheets); and persistent objects into an *object repository*, representing the logical structure of the hypermedia and data about registered users. Finally, the data access module implements an abstract interface for accessing data sources and repository levels.

## 5.2 The Author Module

The author module was designed to efficiently support the author of the hypermedia in the structure definition and content composition phases, as described in Sect. 4. The main features offered by the author module are:

- *Fragment browsing/composing*, with the use of wrappers and comprising the editing of XML metadata associated with fragments
- *Logical structure modeling*, comprising the design (in a visual way) of the logical structure of the hypermedia and the definition of arcs' probabilities. This feature is complemented with a set of utilities regarding the probabilistic structure of the hypermedia (shortest paths, minimum spanning tree, etc.) and with the possibility to validate the graph descriptions of the hypermedia with respect to syntax and semantics, e.g. considering the coherence of probabilities. The author module makes use of persistent representations of the designed structures, thus allowing the reuse of parts of them.

- *Editing of the PDs*, in the form of pure text, graphically as trees, or in a “visual” way. It is possible to create new documents and to edit preexisting ones, with the possibility to preview the final pages.

Finally, since it is fundamental for an author to simulate and validate the probabilistic structure of the hypermedia with respect to the mechanisms that drive the profile assignment decision, the presented architecture includes a *simulation tool*. The tool allows an author to examine in advance the response of the user-modeling component, on the basis of different kinds of users supposed to interact with the system, i.e. to move within its probabilistic structure. By means of the simulation tool the author can analyze the intrinsic properties of the hypermedia calculated from its structure, define a set of *user classes* that describe the behavior of typical users, and analyze the response of the UMC with respect to them. At the end of the simulation the logs of simulated users’ activity and the history of the PDFs can be examined referring to profiles or time. Then, the author can decide to change the overall structure of the hypermedia, the length of the sliding temporal window, or the values of the parameters used to weight the probability density functions [14].

## 6 Concluding Remarks and Future Work

Adaptive hypermedia and the techniques developed behind such systems are increasingly used to add adaptivity to applications in different domains. On the other hand, when AH systems go beyond their original scope, i.e. the application domains for which they were originally designed, different techniques developed in adjacent research fields become increasingly important.

Past and current AH systems are human-centric, and emerging systems enforce the “social” aspects of the interaction between users and AH systems, introducing anthropomorphic, character-based virtual agents. Moreover, ubiquitous computing techniques could be used to monitor and discover the characteristics of the environment hosting the user.

Future Web applications will be characterized by an increased application-to-application interaction, so two questions arise: *how will future Web applications be able to exploit advancements in AH models and systems? How should current AH systems evolve so that they can also interact with applications?* When an adaptive hypermedia (or its software components) has to interact with an application, new problems must be faced, regarding, for example, who is the user, how can his/her/its behaviour be detected, and what kind of adaptation has to be provided.

The Semantic Web [17], is an initiative of the World Wide Web Consortium aiming at augmenting the information available over Internet with semantics. *Web services* provide a systematic and extensible framework for application-to-application interaction by using standard mechanisms to describe, locate, and communicate with online applications. *Adaptive Web services* could embed the adaptive techniques described so far to offer sophisticated services to evolving application requirements. On the other hand, future Web applications including AH could be composed by collections of adaptive Web services.

## References

1. S. Abiteboul, P. Buneman, and D. Suciu. "Data on the Web: From Relations to Semistructured Data and XML". Morgan Kaufmann, 1999.
2. Communications of the ACM, The Adaptive Web (Special Issue on). 45(5), 2002.
3. L. Ardissono, A. Goy, G. Petrone, M. Segnan, and P. Torasso. Tailoring the recommendation of tourist information to heterogeneous user groups. In *Proceedings of OHS-7/SC-3/AH-3*, pages 280–295, 2001.
4. M. Bordegoni, G. Faconti, S. Feiner, M. Maybury, T. Rist, S. Ruggieri, P. Trahanias, and M. Wilson. A standard reference model for intelligent multimedia presentation systems. *Computer Standards and Interfaces*, 18, 1997.
5. J. Borges and M. Levene. Data mining of user navigation patterns. In *Proc. WebKDD'99 Workshop*, 2000.
6. P. De Bra, G.-J. Houben, and H. Wu. AHAM: A Dexter-based reference model for adaptive hypermedia. In K. Tochtermann, J. Westbomke, U.K. Wiil, and J.J. Leggett, editors, *Proceedings of the Conference on Returning to Our Diverse Roots (Hypertext-99)*, pages 147–156, New York, February 21–25 1999. ACM Press.
7. P. Brusilovksi, P. De Bra, and A. Kobsa, editors. *Proc. second workshop on adaptive hypertext and hypermedia*, 1998.
8. P. Brusilovksi, P. De Bra, and A. Kobsa, editors. *Proc. third workshop on adaptive hypertext and hypermedia*, 2001.
9. P. Brusilovksi and R. Conejo, editors. *Proc. International Conference on Adaptive Hypermedia*, 2002.
10. P. Brusilovksi, O. Stock, and C. Strapparava, editors. *Proc. International Conference on Adaptive Hypermedia*, 2000.
11. P. Brusilovski, A. Kobsa, and J. Vassileva. *Adaptive Hypertext and Hypermedia*. Kluwer Academic Publishers, 1998.
12. P. Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3):87–129, 1996.
13. P. Brusilovsky. Adaptive hypermedia. *User Modeling and User-Adapted Interaction, Ten Year Anniversary Issue*, 11(1-2):87–110, 2001.
14. M. Cannataro, A. Cuzzocrea, and A. Pugliese. A probabilistic adaptive hypermedia system. In *Proc. International Conference on Information Technology: Coding and Computing*, 2001.
15. M. Cannataro and A. Pugliese. XAHM: An XML-based adaptive hypermedia model and its implementation. In *Proceedings of OHS-7/SC-3/AH-3*, pages 280–295, 2001.
16. S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks*, 33(1–6):137–157, June 2000.
17. The World Wide Web Consortium. <http://www.w3.org>.
18. P. Fraternali. Tools and approaches for developing data-intensive Web applications: a survey. *ACM Computing Surveys*, 31(3):227–263, 1999.
19. F. Garzotto, P. Paolini, and D. Schwabe. HDM—A model-based approach to hypertext application design. *ACM Transactions on Information Systems*, 11(1):1–26, January 1993.
20. F. Halasz and M. Schwartz. The Dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, 1994.
21. Adaptive Hypertext and Hypermedia home page. <http://www.wis.win.tue.nl/ah/>.
22. A. Kobsa, J. Koenemann, and W. Pohl. Personalized hypermedia presentation techniques for improving online customer relationships. *The Knowledge Engineering Review*, 16:111–155, 2001.



23. D. Schwabe and G. Rossi. An object-oriented approach to Web-based applications design. *Theory and Practice of Object Systems*, 4(4), 1998.
24. H. Wu, E. De Kort, and P. De Bra. Design issues for general-purpose adaptive hypermedia systems. In *Proc. ACM Conference on Hypertext and Hypermedia*, 2001.

---

# Adaptive Web-Based Educational Hypermedia

Paul De Bra, Lora Aroyo, Alexandra Cristea

Eindhoven University of Technology  
Eindhoven, The Netherlands  
{debra, laroyo, acristea}@win.tue.nl

**Summary.** This chapter describes recent and ongoing research to automatically personalize a learning experience through *adaptive educational hypermedia*. The Web has made it possible to give a very large audience access to the same learning material. Rather than offering several versions of learning material about a certain subject, for different types of learners, adaptive educational hypermedia offers personalized learning material without the need to know a detailed classification of users before starting the learning process. We describe different approaches to making a learning experience personalized, all using adaptive hypermedia technology. We include research on authoring for adaptive learning material (the AIMS and MOT projects) and research on modeling adaptive educational applications (the LAOS project). We also cover some of our ongoing work on the AHA! system, which has been used mostly for educational hypermedia but has the potential to be used in very different application areas as well.

## 1 Introduction

Education has been and still is changing dramatically. Until recently education was a well-structured process. Young people went to school (primary school, high school, college or university), following courses taught in classrooms by a physically present teacher. This was a continuous process (interrupted only by vacation) that ended with the highest degree the learner would ever obtain. An evolution that we have seen in this process is a shift from purely oral communication, through note taking by the learner and later note preparation by the teacher, to the publication of textbooks that (when they are very good) are beginning to make the physical presence of teachers and learners in a classroom superfluous. Several additional changes are occurring simultaneously, characterized by the three A's *anyplace, anytime, anyhow*.<sup>1</sup> Learners no longer want to go through one multi-year course program leading to a 'final' degree in the shortest possible time. They want to combine study and work, and as a result are not available for a teaching/learning session in a classroom at scheduled times.

---

<sup>1</sup> The three A's are mentioned very often in presentations, but we have been unable to find out who first came up with them, hence the missing reference.

The education process must allow learners to study at times when their work allows. The process must not require the students to physically travel to an institute in order to be able to learn. The process must also accommodate different ways of learning, provided these ways lead to the same knowledge and skills.

The Internet provides a technological basis for making the ‘new education’ possible, because it is available *anyplace* and *anytime*. However, the Internet does not automatically make the teacher available *anyplace* and *anytime*. We need a means to make the teaching/learning process independent of the physical availability of the teacher. For very interactive teaching processes many researchers have tried to use computer technology to create teaching systems that try to *simulate* the intelligence of the teacher. (See, e.g. the many publications in the field of intelligent tutoring systems, or ITS.) A common problem with this approach is that many ITS fail to give the impression of being intelligent. Nonetheless there have been a number of successful ITS like ELM-ART [29, 30] and SQL-Tutor [25, 26]. A different approach is to rely on the intelligence of the learner to select the information and assignments needed to master the subject of a course. This approach puts the emphasis on making information available to the learner. A first step in this direction was the creation of good textbooks. A second step was the transformation of such textbooks into hypermedia form (and putting them on the Web). This step gave learners the freedom to select information and to choose their own reading order. A third step is to make the hypermedia textbook ‘dynamic’ in the sense that it changes itself (possibly in an invisible way) to accommodate the reading order chosen by the learner. The result of this step is what we call *adaptive educational hypermedia*, the subject of this chapter.

Adaptive educational hypermedia provides adaptation regarding the three C’s *connectivity*, *content* and *culture*.<sup>2</sup> Connectivity is what distinguishes hypermedia from (text)books: information items are linked together in many ways, offering learners many possible ways to navigate through the information. Adaptation means that the system can offer guidance by making distinctions between links, based on how ‘appropriate’ or ‘relevant’ links are for a given learner. Content adaptation enables the system to offer additional notes to learners who appear to need them (thus compensating for information the learner missed in his/her navigation path), or to omit information the learner appears not to need (because it was acquired elsewhere). Culture represents that learners have different preferences regarding *how* they are able to learn. A single adaptive educational hypermedia application must accommodate different learning styles. Adaptation in the three C’s realizes the most important of the three A’s: *anyhow*.

In the next section we give a sketch of the architecture of adaptive Web-based educational hypermedia. We specifically focus on what a system needs in order to be able to provide adaptation. We refer to some well-known adaptive educational hypermedia systems (some also predating the Web) to illustrate the use of the concepts covered by the architecture. After that we look at some current developments and research ideas that may shape future Web-based educational systems.

<sup>2</sup> Again, we do not know the origin of the three C’s.

## 2 Adaptive Educational Hypermedia Architecture and Systems

In this section we describe the main architectural elements of adaptive educational hypermedia applications. We do not intend to give a complete reference architecture like AHAM [9, 31] or the Munich Model [23, 24]. We concentrate on the *user model*, a system representation of how the learner relates to the conceptual structure of the application, the way in which the system *updates* this user model, and the way in which the state of the user model affects how the system *adapts* to the learner.

### 2.1 Web-Based Adaptive Systems

Adaptive educational systems predate the Web. The advantages of Web-based systems (mostly the availability anytime and anywhere) may make us forget the problems of Web-based systems. A non-Web-based system can have a tight integration of the user interface and the underlying functionality. Every action of the user can be recorded: every mouse movement, the window size and placement, scrolling, etc. This is not possible on the Web. Web-based systems assume a client-server approach in which the client is fairly dumb. It presents information it receives from the server, and it sends user requests to the server. These consist of either a GET request for another page (as a result of clicking on a link anchor) or a POST request that results from filling out a form, perhaps consisting of answers to questions of an evaluation test, or perhaps specifying a search for certain terms. The ‘intelligence’ resides completely on the server side. This has the advantage that the same learner can interact with the educational application from any browser anywhere on the Internet. When the server receives a request (which it can associate to a particular learner) it performs the following series of actions:

1. The system first retrieves the user model from permanent storage (file system or database). It may keep that user model in main memory in between requests, for performance reasons.
2. The system must have a representation of the conceptual structure of the application’s knowledge domain. This structure is normally retrieved from permanent storage (and then kept in memory) and thus is not integrated into the system.
3. Part of the conceptual structure is a definition of how the user’s request and the current state of the user model must be used to calculate a new state of the user model. The user’s request reflects some change in the user’s knowledge state, and the system must make the user model reflect that change.
4. The user’s request results in a response that has to be sent back to the browser. This can be an information page or a dynamically generated page giving feedback on a submitted form (test or search). Again, the conceptual structure contains a definition of how that presentation is influenced by the state of the user model. This *adaptation* process normally uses the *new* state of the user model.
5. When the adapted response is sent to the browser the new (updated) user model is saved. Waiting to save the user model until after sending the response to the browser improves the system’s response time and enables the system to undo

the user model changes (by not saving them) in case sending the response to the browser fails.

In the following subsections we discuss the different parts of an adaptive educational application. Although existing systems vary widely, they do share some common parts.

## 2.2 User Model

An educational application is used to learn about certain subjects. The information domain of the application can be divided into such subjects, topics or concepts. In the following we shall always use the term *concept*. A *user model* is used to represent how the user *relates* to these concepts. In adaptive applications this notion of *relates* can mean many things, but in education applications it typically represents the learner's *knowledge* about the concepts. A user model that models the learner's knowledge about concepts through a *knowledge value* (or knowledge level) per concept is called an *overlay model*.

There are two ways of modeling knowledge of concepts: *coarse grained* and *fine grained*, and three ways to model knowledge values: *binary*, *enumerated* and (*pseudo*) *continuous*.

- With *coarse-grained* modeling of knowledge we mean that the knowledge domain is modeled using only a few 'broad' concepts. Learner actions (like accessing/reading pages and performing assignments or tests) result in changes to the knowledge value for some concept(s). In order to model the gradual increase of knowledge about a broad concept a value domain is needed with many different values. Real numbers between 0 and 1, or integers between 0 and 100 (to mean a percentage) are examples of value domains that can be used.
- With *fine-grained* modeling of knowledge we mean that small items like pages or perhaps even fragments of pages or single media objects (like images) are represented as concepts in the user model. When there are many small concepts the learner's knowledge can be represented using simple values about these concepts. A binary value, like *known* or *not known*, can even be sufficient and was used in early versions of the AHA! system [8]. The system becomes more expressive if a few more values are used, like *well known*, *known*, *partly known* and *not known* (used with slightly different terms in Interbook [12]). It does not make much sense to describe the knowledge about small concepts much more in detail (like with a percentage) because the system has very little information to base such a detailed judgement on. (For example, how would you decide whether a user who reads a page has 73 or 74% knowledge of that page?)

Coarse- and fine-grained user modeling both have advantages. In a system that keeps track of the learner's knowledge of a few broad topics (using a detailed scale of values), one can easily generate an overview of the knowledge that is easy to understand. Also, performing adaptation to the learner's knowledge can be done using these few knowledge variables that have a clear meaning. As a result the adaptive behavior of

the educational application is predictable and easy to understand and explain. On the other hand, modeling the learner's knowledge in detail, using many small concepts, perhaps with only a crude representation of knowledge, enables a system to adapt to this detailed knowledge. If the learner does not know a particular term used on a page, the system can add a short explanation of that term, or it can replace the term by a nontechnical equivalent. Ideally, an adaptive educational system would combine the strong points of both knowledge representations. The system should represent detailed and global knowledge. This implies that (if the same value domain is used for all concepts) a rich value domain is used for small concepts as well as broad concepts, which is in some sense overkill.

In an educational application the user model also contains information about the learner, independent of the specific subject domain of the application. An example of such information is a *learning style*. Some simple aspects of how a learner likes concepts to be presented can be used by an adaptive system. Some learners like to see a few examples before the definition or description of a concept, whereas others prefer to see the definition first and some examples later. Some learners prefer to learn just a bit about a concept and discover the rest through assignments, whereas others prefer to study everything and only perform tests or assignments to verify their knowledge (and not to increase it).

Some educational applications also support additional information about the learner's intentions and goals: the *task model*. Often the learner has to perform tasks only within a specific subpart of the application and not in the whole subject domain. The task model allows for splitting the subject domain into separate complementary subjects (or topics) and assigns the necessary corresponding tasks in order to achieve a specific learning goal. This way, rather than learning everything, the learner can navigate and focus on a specific subset of the subject domain according to his/her course task. That subset does not necessarily have to correspond to a chapter structure of a textbook. Within educational systems with concept-oriented domain descriptions, often the most natural way of describing a task would be by using the concepts of the user model. It is then also an *overlay model*, and it is typically a fine grained model. The relevance of the concepts for the task is used to determine which parts the learner should study for this task. The task model can also be used to improve a search facility or a graphical concept map, by adding additional terms to a search string or filtering out the nonrelevant concepts from the map. Both techniques have been used in AIMS [1, 2], for instance.

### 2.3 Domain and Adaptation Model

Adaptive educational hypermedia applications deal with a subject domain. At an abstract level that domain can be described using (high- or low-level) *concepts*, the same concepts that are used in the user model (which explains the term *overlay model*). The application also consists of *pages*, which are often considered as low-level concepts.

A commonly used modeling approach for the conceptual structure of an educational application is the use of a *concept hierarchy*. This follows the typical structure of

a textbook, consisting of chapters, sections, subsections and paragraphs. The system can update the user model by considering the propagation of knowledge from pages to subsections to sections to chapters. By considering high- and low-level concepts it becomes easier to specify adaptation that depends on details as well as adaptation that depends on the knowledge of a whole chapter.

The concept hierarchy describes the structure of the application domain, but not the way in which the learner should or could navigate through that domain. Considering only the hierarchy, the learner could start descending to the page level and start studying in the last as well as the first chapter. In fact, there need not be a 'first' versus 'last' chapter as the hierarchy can be unordered. Furthermore, the *hypermedia* aspect is realized through associative or cross-reference links throughout the application. These links connect pages in completely (structurally) different parts of the application, based on the meaning of the pages. Following arbitrary links may lead to *orientation* and *comprehension* problems:

- Following arbitrary links through any hypermedia structure always causes a risk of not knowing where you are. This problem is independent of whether the application is of an educational nature or not. Solutions usually involve displaying part of the structure surrounding the 'current' page, possibly adapted to the user in some way. This is called 'map adaptation' in Brusilovsky's taxonomy [10, 11].
- The comprehension problem is typical for educational hypermedia: it is likely that a user will be able to follow a path that leads to pages using technical terms without passing by pages that explain these terms. This problem can be tackled in two ways: by blocking such paths or by adding the explanation when needed.

Adaptation in educational hypermedia is most often based on what are called *prerequisite relationships* between concepts or pages. 'A is a prerequisite for B' means that the learner should 'know' A before studying B. Defining prerequisite relationships between high-level concepts may be hard for a teacher. But prerequisite relationships that indicate that a page containing an explanation of a term should be read before going to a page that uses that term are easy to identify. The adaptation offered by the system relieves the author from the task of ensuring that all possible navigation paths obey the prerequisites. The system can enforce the use of valid navigation paths or compensate for the missing knowledge. The way in which the system adapts the presentation of information and links by using the prerequisites is defined in what the AHAM model [9] calls the *adaptation model* (AM).

- In Interbook [12] the prerequisite relationships are used to provide *annotation* to the links, in the form of colored balls. When the learner is ready to read a page that offers new (not previously studied) information a green ball is shown next to the links (actually link anchors) to that page. If the learner is not ready, meaning that some prerequisite knowledge is missing, the annotation becomes a red ball. Interbook offers more adaptive features that also use this color metaphor. A 'teach me' button on a page (with unsatisfied prerequisites) generates a list of links to pages that need to be studied in order to acquire all the prerequisite knowledge for

the current page. In this list the colored balls appear again, to indicate which pages to study first (the ones with green balls). There is always a reading order that only contains ‘recommended’ pages, because the structure of prerequisite relationships must be *acyclic*. The entire adaptive behavior of Interbook is ‘hard-wired’ into the system. Interbook thus has a fixed, built-in adaptation model.

- In AHA! [8] the standard behavior is that of *link hiding*. Links normally appear in blue (unvisited) or purple (visited). But when a link leads to a page for which the learner is missing some prerequisite knowledge, the link anchor is displayed in black and is not underlined. The anchor is therefore indistinguishable from plain text and effectively ‘hidden’. AHA! does not really use prerequisite relationships but a powerful event–condition–action rule language that is capable of expressing many different types of concept relationships, including prerequisites, of course.
- Brusilovsky’s (original) taxonomy [10] mentions three other types of *adaptive navigation support*: *direct guidance*, in which the learner is shown a ‘next’ button or some similar indication of the ‘best’ next page to read; *adaptive sorting of links*, meaning that a list of links is sorted from most relevant to least relevant, for this particular user; and *map adaptation*, which we explained earlier. All techniques for adaptive navigation support try to point out the recommended link(s) to the user.
- A completely different way to deal with prerequisites is to *compensate* for the missing knowledge, by adding a *prerequisite explanation* to a page for which the learner is not really ready. This technique is used in the AHA!-based hypermedia course at the Eindhoven University of Technology.<sup>3</sup> It is a special case of a more general technique of *conditional inclusion of fragments*, implemented in AHA! as well as in other systems, like C-Book [21], for instance. In C-Book the technique is also used to offer *additional* or *comparative* explanations. Learners who are familiar with Pascal are given comparisons between C constructs they are learning and similar Pascal constructs they already know. These comparisons are useless to non-Pascal programmers and therefore are only shown to Pascal programmers. In the *stretchtext* technique the recommended fragments are shown to the user, but the hidden fragments are accessible through (small iconic) links or buttons. Using this technique all information is available to everyone.

The techniques mentioned above are the most commonly used techniques, but there are many more possibilities. In the AHAM model [9] one can define arbitrary concept relationship types and associate them with user model updates and adaptation. An example is the *inhibitor* type, which can be viewed as the opposite of a prerequisite. An online course may, for instance, contain several alternative descriptions of the same concept. Once the learner has studied the concept through one of these descriptions, access to the alternative descriptions may be inhibited. Similarly, a page may contain some explanation that is automatically omitted when the learner already has some specific knowledge.

Figure 1 shows an authoring interface developed This hierarchy is used to indicate how pages contribute knowledge for the AHA! system. On the left it shows the concept

<sup>3</sup> See <http://wwwis.win.tue.nl/2L690/>.



hierarchy, to higher-level concepts. When (like in the example) there are two pages contributing knowledge to the concept ‘beer’ each page contributes 50%. And since there are two concepts (‘beer’ and ‘chocolate’) contributing knowledge to the concept ‘belgium’, each contributes half again. As a result, reading a page contributes 25% knowledge to ‘belgium’. These values can be altered, much in the same way as in the system described in [19]. On the right the concepts (dragged from the left) can be connected using concept relationships of different types. Here the solid arrows mean ‘interest relationships’, and the dashed arrows are prerequisites. They indicate that one first needs to read about a ‘plain’ beer (Stella) before going on to a more local, special beer (De Koninck), and that one needs to read about a fairly common chocolate (Meurisse) before going to a real Belgian specialty brand (Callebaut).

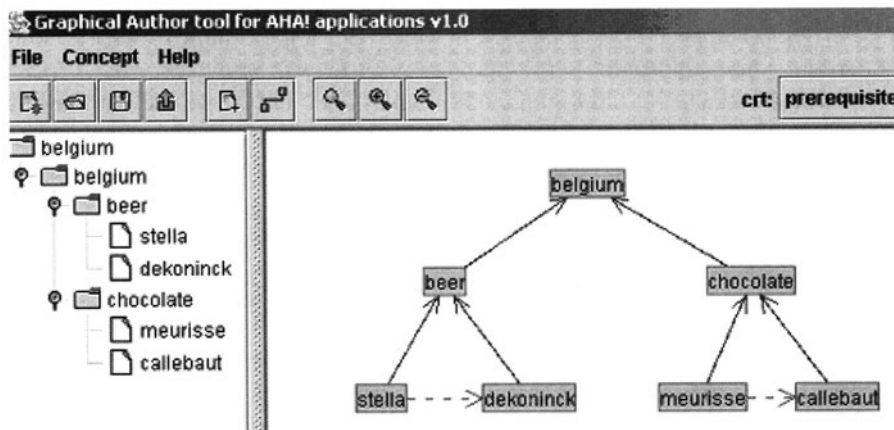


Fig. 1. AHA! authoring tool for concept hierarchy and concept relationships

Whereas an interface like Fig. 1 defines the application domain at an abstract level and expresses some desired adaptation aspects, it does not define the desired behavior of the system in terms of the user model updates and the link and content adaptation. Indeed, as shown above, there are different ways to perform adaptation based on prerequisite relationships. In most adaptive educational platforms that ‘performance’ is completely hard-coded. But in AHA! it is defined through a set of event–condition–action rules that define this behavior. This approach was taken to reflect the low-level functionality suggested by the AHAM model, and is described in Section 3.

The common behavior of adaptive educational systems is to *first* update the user model, taking into account the knowledge that will be gained by reading the requested page. In a *second* phase the page is generated, conditionally including fragments that depend on prerequisite or other concept relationships, and annotating or hiding links based on these relationships. It is important to note this order of actions. Intuitively, one

would expect a page to be presented based on the user model state *prior to* presenting the page. This makes perfect sense because if the user is missing some prerequisite knowledge that can be compensated for by inserting an explanation, the system must look at the user model state before generating the page in order to conclude that the prerequisite explanation must be inserted. However, if the page to be presented contains prerequisite knowledge for other pages to which the page also contains links, one wishes these links to be shown as 'recommended'. Since there is no (easy) way to change the presentation of a page once it is shown to the user, these links cannot first be shown as 'not recommended' and after some reading time changed to 'recommended'. Hence it is necessary to show the links as 'recommended' as soon as the user retrieves the page. It is clearly a limitation or drawback to first update the user model and then generate the presentation, but for the inclusion of prerequisite explanations it turns out not to be a big problem as the prerequisite explanation normally does not replace the full explanation. In other words, the page with the prerequisite explanation included should not generate enough knowledge about the missing prerequisite concept to make the prerequisite explanation not be included (or at least not the first time this page is accessed). It takes careful system design to ensure that the counterintuitive order (of first updating the user model and then generating the presentation) indeed produces the expected presentation. In Section 3 we show how this is done in AHA! using its event-condition-action rules.

## 2.4 Assessment

It is an illusion that an adaptive educational hypermedia system can accurately assess a learner's knowledge by observing which pages the learner reads. The technique can be refined, by observing the reading time, by taking into account which fragments are included in the pages the learner reads, and perhaps even by performing eye-tracking to check which parts of a page the learner really reads. All these more or less intrusive ways of monitoring the learner fail to grasp what really goes on in the learner's mind. Therefore many adaptive educational hypermedia applications resort to a different technique for measuring the learner's knowledge: tests.

Multiple-choice tests are most popular because, although they are difficult to create, they are easy to evaluate. In theory a multiple-choice test can be implemented using 'plain' adaptive educational hypermedia. A page can be shown that contains a question and several answers. Each answer is a link to another page that explains why the answer is right or wrong. Accessing these pages (by clicking on the answer) performs a user model update just like accessing 'normal' pages. The difference is that when reading information pages knowledge can only increase, whereas clicking on 'answer' pages may cause the knowledge value to decrease. In AHA! [8], for instance, every page access can cause knowledge increases as well as decreases.

Most systems (including AHA!) offer a separate module for (multiple-choice) tests. In the AHA!-based hypermedia course, the learner is first guided to three introductory chapters. Each chapter ends with a multiple-choice test. Access to (that is, unhiding of the links to) the advanced chapters is granted after completing the three multiple-choice tests, not after reading most or all the pages of these chapters.

At the end of the course another test, containing 15 (randomly selected) questions, is included before access is granted to the final assignment. Initially the test contained a fixed sequence of 20 questions. But after having this course available for some time, we discovered a drawback of the *anytime* (and perhaps *anyhow*) aspects of Web-based learning: more and more students achieved a perfect score on the difficult final test. It turned out that the correct sequence of answers was being communicated among students. This taught us that in an online course the student assessment needs to use a different test for each student (or at least enough randomness that communicating the answers becomes difficult).

Multiple-choice tests can be very useful in initial tests to determine the entry-level of the learner, and also to determine the level at different points in time. In a (Spanish) Linux course, Romero [28] used a modified AHA! application with initial and final multiple-choice tests. Log files from this course were used (with association rule mining) to find out potential problems in the course. If information pages and follow-up tests are not well matched, this will show up in the test results. For example, if the test is too straightforward after reading the pages then virtually all the students will answer the questions correctly. If the test is too difficult, e.g. because the information needed to answer the questions cannot be found in the preceding information pages then many students will give incorrect answers. Likewise, if the information is there but either it is wrong or the answer indicated as the correct one is wrong, then many students will get a low score as well.

## 2.5 Learning Objects and Learning Standards

Since adaptive educational hypermedia applications are centered around the notion of *concept*, one may wonder whether the architecture compares to the use of *learning objects*, as used in different standards or standards initiatives, such as the LOM (Learning Objects Metadata) standard from IEEE Learning Technology Standards Committee (see <http://ltsc.ieee.org/>), or the SCORM (Shareable Courseware Object Reference Model) from the US Government Advanced Distributed Learning (ADL) Initiative, or IMS (Instructional Management System) from the Global Learning Consortium. A comprehensive page about such initiatives can be found at <http://www.learnativity.com/> (under the heading ‘standards and learning objects’, accessed on 1 July, 2003). The focus of these initiatives is on distribution and sharing. In order for teachers to construct courses from material gathered from different sources, a clear description is needed of what that material entails. Therefore, each piece of instructional material, called a ‘learning object’, is tagged with a significant amount of metadata. Just like concepts, learning objects can be large or small, entire courses or lectures as well as a single illustration, application or applet, or a paragraph of text.

Although an author can build a course from learning objects that reside on different servers (they need not be copied to a single location in order to use them, and in the case of server applications they often even cannot be copied), it is difficult to use the learning objects and their metadata to perform *adaptation*.

We expect that sometime in the near future the worlds of adaptive educational applications and of distributed learning environments will come together, but at the moment of this writing they are still too far apart to consider the use of these learning standards in *adaptive* educational systems.

### 3 AHA! The Adaptive Hypermedia Architecture

In 1994 we started offering a course on hypermedia at the Eindhoven University of Technology. This course went through several generations and is now available as <http://wwwis.win.tue.nl/2L690/>. In 1996 we started adding adaptive functionality to the course; hence AHA! was born. The first version was based on CGI scripts written in C. The browsers used at that time lacked the ability to ‘play’ with link colors through style sheets, a feature we currently use to create the *good*, *neutral* and *bad* link colors (which are blue, purple and black by default). Therefore, instead of simply ‘hiding’ unrecommended links, the anchor (<A>) tags were removed, thus disabling the use of unrecommended links as well as hiding them. Conditional inclusion of fragments was done through C-preprocessor constructs (#if statements). See [13] for details.

In the next version [7] the C-preprocessor constructs were replaced by HTML comments. Link removal/disabling was still used as in the initial version. The user model in these early versions consists of one concept per page, with a binary value, indicating *known* or *unknown*.

#### 3.1 AHA! Version 1.0

The first version of AHA! used outside the Eindhoven University of Technology is called version 1.0. This is the version used by Cini et. al. [14] to measure the ‘presentation adaptivity degree’ in adaptive applications, and by Romero et. al. [28] to perform association rule mining to help authors of adaptive courses spot anomalies in their courses. AHA! 1.0 supported the main features described in Section 2 as necessary in adaptive educational applications. The main features of AHA! 1.0 are the following:

- The user model in AHA! 1.0 consists of one *concept* for every page, and possibly many more *abstract concepts*, not associated with a page. For every concept the learner’s knowledge is stored as a percentage (an integer value between 0 and 100). This allows more values than needed for pages, but is mostly useful for keeping track of the user’s knowledge about larger concepts (like whole sections or chapters of a course).
- For every concept there is a *rule* that defines how a change in the knowledge of the concept influences the knowledge of other concepts. As an example, when the rule for a concept A contains:

B: +40 C: 30 D: -50

it means that when the knowledge of A is increased by X the knowledge of B will be increased by 40% of X, the knowledge of C will be set to 30, and the knowledge of D will be decreased by 50% of X. Increases or decreases, however, can never make a value lower than 0 or higher than 100. The change X to the knowledge value of A can be positive or negative.

- For every page there is also a *requirement* that defines the prerequisites. This requirement may look like:

E>50 and F<30

meaning that this page is ‘recommended’ when the knowledge of concept E exceeds 50 and the knowledge of F is lower than 30. It is thus possible to express *prerequisite relationships* as well as *inhibitors*. When the learner visits a page the knowledge of that page becomes 100 if the page was recommended. If the page was not recommended the knowledge becomes 35 or the previous value, whichever is higher. AHA! thus expresses that reading a page generates partial or full knowledge of that page depending on the satisfaction of prerequisites.

- Because (like in the ‘beer’ example of Fig. 1) knowledge often needs to be propagated to a high-level concept, the knowledge update rules may trigger each other. The rule for Stella would read:

beer:+50

and the rule for beer would read

belgium:+50

When reading about Stella (for the first time) the knowledge of Stella jumps from 0 to 100. This causes the knowledge of beer to be incremented by 50 (50% of 100) and that of belgium by 25% (50% of 50). Through these rules one can easily describe the knowledge propagation through a concept hierarchy. One should keep in mind that because of the use of integer values there may be rounding (truncation) errors, causing the knowledge value of a high-level concept to not reach 100 when one would expect it to. (In [31] an example is given to illustrate that the user model updates can depend on the execution order of the rules, because of the truncation in integer arithmetic.)

- An inherent danger of rule propagation is the occurrence of infinite loops. Indeed, rules can interact with each other, generating an infinite sequence of knowledge value updates that go up and down. In order to prevent such loops, AHA! 1.0 only performs propagation on monotonic relative updates (the ones with a ‘+’ sign in the rules). Should there still be a loop in the rules, the effect can only be that several knowledge values are repeatedly updated in the same direction (all increased or all decreased). Such a loop must end when all values become 100 or 0. Because there are only a finite number of concepts and only a finite (101) number of possible values for each concept, the loop must end after a finite number of steps.
- AHA! uses the cascading style sheet mechanism to tell the browser how to display links. AHA! tells the browser not to underline links, so that black link text is indis-

tinguishable from plain text (without a link anchor). The default link adaptation mechanism in AHA! is that of *link hiding*: recommended links are blue (unvisited) or purple (visited), and unrecommended links are black. The user can change this color scheme, making the unrecommended links more visible. According to Brusilovsky's taxonomy [10], AHA! then supports *link annotation*.

- A special 'if' tag is used in pages to indicate conditionally included fragments. The conditions for fragment inclusion are the same as for page recommendation, and can therefore express prerequisites as well as inhibitors.
- AHA! contains some special features, like a module for generating and evaluating multiple-choice tests, an optional page header including the number of pages read or still to be read, links to a list of read pages and a list of unread pages, and an invisible Java applet that makes the server record the reading time for each page.

Modeling the knowledge structure of an educational application is easy in AHA!. Adaptation based on prerequisite relationships is also fairly straightforward.

### 3.2 AHA! Version 2.0: More Adaptation Flexibility

In AHA! 1.0 for every user model concept there is a single integer value between 0 and 100. In the previous section we implicitly assumed that a concept means a concept from the application domain and that the value means the (system's idea of the) user's knowledge about that concept. However, there is nothing in the AHA! system to support that assumption. It is all in the eyes of the beholder.

Concepts in AHA! are just variables that can be manipulated through rules that are triggered by page accesses. This can be convenient to add adaptation to parts of the information presentation that are not related to knowledge. AHA! does not enforce a specific presentation style or look and feel, and is in that respect very different from most other adaptive educational systems. We have created an adaptive course with HTML frames, using a 'left' frame to show navigation support in the form of a Microsoft Windows Explorer-like hierarchical menu, and a 'right' frame to show the information pages. We have used adaptation to make submenus open and close automatically depending on the location of the 'current' page in the concept hierarchy. When following a cross-reference link, the learner jumps to a completely different position in the concept hierarchy. The Explorer-like navigation aid would follow this jump and show the context of the new page.

AHA! 2.0 [6] makes it a lot easier to create ad hoc adaptive behavior. Concepts can have arbitrarily many (named) attributes with Boolean, integer or string values. Typical attributes are:

- *Access*: This is a Boolean attribute with a system-defined meaning. When a page is accessed, the access attribute of the concept(s) associated with this page becomes true. This triggers the execution of a set of *event-condition-action rules* associated with the attribute. A typical rule for the access attribute would be to increment the *knowledge* of the concept with an amount that depends on the *suitability* of the

concept. The access attribute is volatile: its value is not stored in the user model.

- *Knowledge*: This attribute represents the user's knowledge about the concept. There are typically some rules associated with this attribute, to 'propagate' knowledge to higher-level concepts. This is similar to the way the updates worked in AHA! 1.0. Changes to attribute values (like knowledge) trigger the rules associated with these attributes (of the changed concepts). This 'propagation' can be turned on or off for each rule.
- *Interest*: This attribute represents the user's interest in the concept. It may depend, for instance, on the user's task or goal. AHA! can use the attribute value to guide the link annotation or hiding, in order to provide the user with links to pages that correspond to his/her interest.
- *Suitability*: This attribute can be used to remember whether the concept is suitable for the user. This can, e.g., mean that the user has all the required prerequisite knowledge. Using the suitability attribute expressions for the conditional inclusion of fragments can also be simplified.
- *Visited*: This attribute remembers whether the user visited this concept (usually a page). It is used by AHA! to choose between the *good* and *neutral* link colors, which are blue and purple by default, to mimic the standard browser behavior.

Using the above-mentioned attributes and carefully chosen event-condition-action rules the user modeling and adaptation possibilities in AHA! are very powerful. As an example, it is easy to increment a knowledge value by a small amount each time a page is visited. The conditional inclusion of fragment (containing a prerequisite explanation) can be done based on that small amount. If knowledge is initially 0 and is incremented by 10 on each visit, a fragment with expression 'knowledge < 20' will be shown the first time, and hidden from the second visit on. On the first visit the knowledge value goes from 0 to 10 *before* the page generated. The conditional inclusion of fragments thus uses the updated user model values. On the second visit the knowledge value becomes 20, and the fragment is no longer included. The knowledge value can be updated through different pages as well, so a prerequisite explanation can be shown on the first out of a set of many pages on a certain concept, etc.

An concept editor (Java applet based) hides the underlying XML syntax of the event-condition-action rules from the author. Two examples of rules (in an arbitrary syntax not used by AHA!) are given below.

**E:** stella.access

**C:** stella.suitability > 50

**A:** stella.knowledge = 100

This rule is triggered when the *access* attribute of *stella* changes, which is the case when the page about Stella is accessed. The rule is executed when the *suitability* of Stella is high enough, and the action is to set the *knowledge* attribute to 100. A second rule could then be:

**E:** stella.knowledge

**C:** beer.knowledge < 100

**A:** beer.knowledge + = 0.5 \* stella.knowledge

This rule is triggered by a knowledge change of Stella and adds 50% of that change to the knowledge of beer if that knowledge is still less than 100. (The underscore \_ is used to indicate that the change of the attribute value is used instead of the value itself. It is easy to say that the following rule then performs the 25% increase in knowledge about Belgium, as described earlier:

**E:** beer.knowledge

**C:** belgium.knowledge < 100

**A:** belgium.knowledge+ = 0.5\*.beer.knowledge

### 3.3 Beyond AHA! 2.0: Better Authoring Support

The user modeling and adaptation flexibility of AHA! 2.0 comes at a price: the complexity of authoring. AHA! does not know how to deal with prerequisite relationships. These have to be encoded using attributes and event-condition-action rules. AHA! also does not know about learning styles. They too have to be translated to rules.

Figure 1 shows a first attempt to enable higher-level authoring in AHA!. The available *types* of concept relationships are defined by a system designer, as well as their translation to the AHA! event-condition-action rules. The author simply chooses concept relationship types from a list.

The conditional inclusion of fragments is also being changed in AHA!. In version 2.0 the fragments are parts of the (XHTML) pages, whereas in the upcoming version they will be separate objects. This has the advantage that a fragment that may appear on different pages can be stored just once.

What AHA! continues to lack is support for yet higher level aspects such as task support and adaptation to learning styles. In the next section we present some new research into adaptive educational hypermedia authoring, and the support for tasks, styles and goals.

## 4 New Adaptive Hypermedia Authoring Techniques and Implementations

The potential benefits of adaptive educational hypermedia for educational applications should be clear from the previous sections. However, surprisingly, we do not find adaptive hypermedia so wide spread and widely accepted as we would expect. To find an explanation for this fact, we start by looking at the weak points in present adaptive hypermedia systems (AHS): the difficulties in authoring and the lack of support for partial learning, i.e. learning just one subject of a course, or searching for the information needed for one specific task. We proceed by searching for methods for solving the authoring problems and for task-based searching. Sections 4.1 and 4.2 describe ongoing research in task support, considering both the learner's tasks and the authoring tasks. Section 4.3 describes an extension of the AHAM model, considering high-level adaptation to goals and constraints as well as low-level ad hoc adaptation.



Section 4.4 shows how these ideas are being experimented with in an experimental authoring tool.

#### **4.1 Authoring Support Framework for Adaptive (Knowledge-based) Courseware**

In the previous sections we saw examples of AH authoring activities as they were performed within the AHA! system. As concluded above, different AHS share common modules with respect to defining domain, adaptation, user and task models. Subsequently, the authoring activities for instantiating these models within various AHS systems also share common functionality. AIMS is an example of an intelligent courseware tool that follows an instructional model (analogous to the adaptation model in AHA!) over a concept domain definition and overlaid user and task models. This way it provides support for instructors to build online courses as well as for learners to identify information necessary for performing course tasks (e.g. course assignments) [2]. The system can be used stand-alone (for example, as an extension to a traditional or online distance course) or can be integrated in a larger electronic learning/training/work environment that allows the users to perform open learning tasks in a specific subject domain. In both cases it provides the user with immediate, online access to a broad range of structured information and with domain-related help in the context of work, thus supporting more efficient task performance. In Fig. 2 the student task-oriented search and browsing environment is presented, which employs the above approach. The authoring environment is shown in Fig. 3.

To ensure this we employ the AHS approach for user and domain models and introduce the resource model for knowledge classification and indexing based on conceptualization of the course material and subject domain. The instructional model here provides the main set of rules in order to link the above models and produce the desired instructional result. In the line of recent advances in the research related to the Semantic Web and ontologies [4], AIMS focuses on using ontologies for defining rules and structures for the subject domain, user model construction, course structure building and resource managements. The goal here is twofold. On one hand, an ontology offers a means to define vocabulary, structure and constraints for expressing metadata about learning resources. It also offers formal semantics for the primitives defined in the ontological structures and thus more efficient reasoning, which is useful for the support of both the learning and the authoring processes. On the other hand, ontologies offer a solution for shareable and reusable knowledge bases among different courses and authors (instructors), which is imperative in educational environments. By reaching these goals we provide a basis for more efficient task-oriented information retrieval and adaptation to the learner status [3], as well as methods to fill the deep conceptual gap between authoring systems and the authors.

As we have seen in AHA! as well as in AIMS, building instructional structures, adaptive navigation and content presentation appears to be an extremely difficult and time-consuming task. The successful implementation of the suggested approaches turns out to depend crucially on the ontological engineering for providing means for sharing and reusing instructional knowledge. Thus, the key focus here becomes the

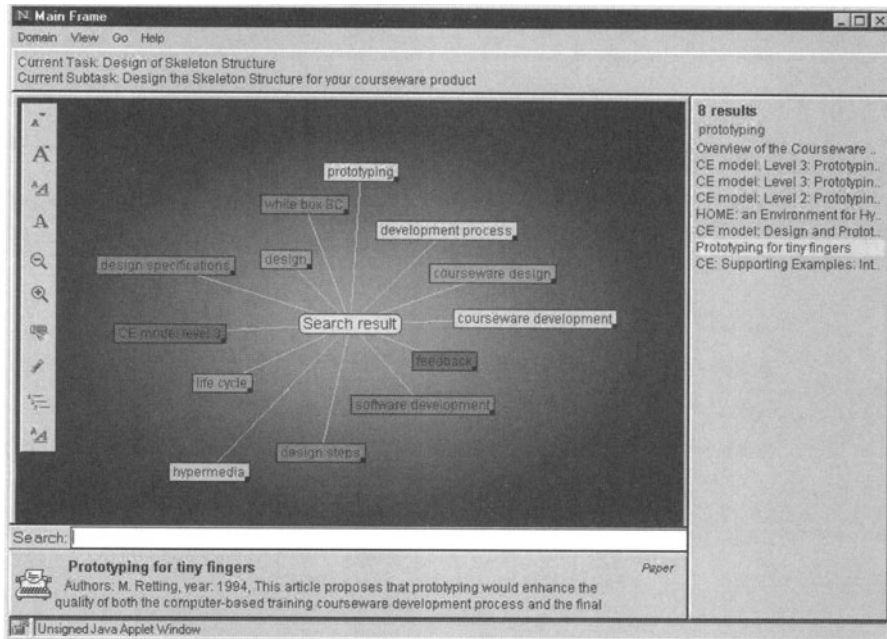


Fig. 2. AIMS student information search and domain navigation environment

authoring process and the formalization of the knowledge about it, which can be employed within an intelligent authoring tool. In this context we explore the construction of and the interoperability between the various ontological structures for domain and instructional modeling and the modeling of the entire authoring process. In this we decompose the authoring process into (1) a set of generic atomic tasks underlying (2) higher-level authoring activities related to the subject domain ontology, dynamic course structure and resource management metadata. This is the first step toward building a metaontology of courseware authoring functional concepts (generic authoring tasks), in the same sense as introduced by [22]. The intention is to use it as a basis for defining a corresponding set of tools to support the main phases of ontology engineering for courseware authoring.

## 4.2 Authoring Task Ontology

The Authoring Task Ontology (ATO) is based on the notion of 'task ontology' defined by Mizoguchi et. al. [27]. It provides operational semantics to the authoring process by specifying, in a domain-independent way, all authoring activities, sub-activities, goals and stages based on a common system of vocabulary. This way ATO enables the authoring tools to guide, support, navigate and check the consistency and completeness of the authoring activities at all levels of complexity. In the construction of ATO we consider specification of top-level authoring activities, low-level authoring functions,

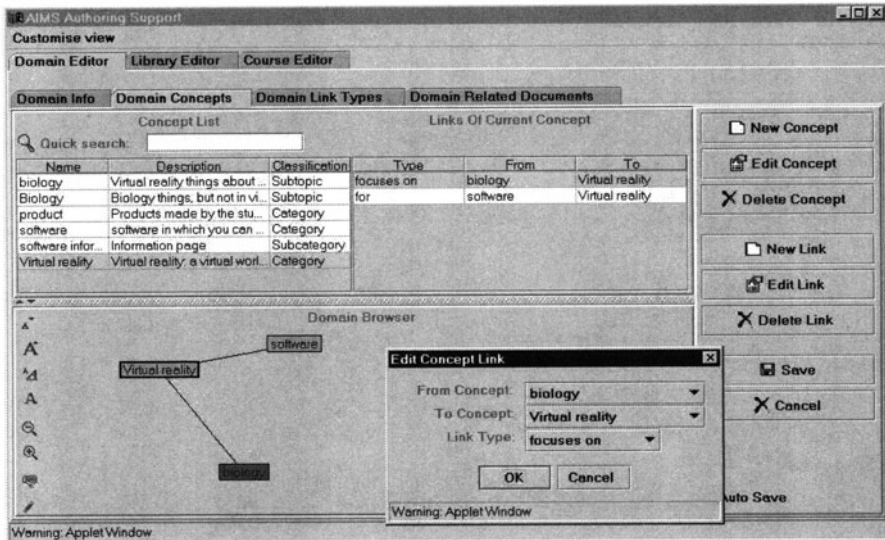


Fig. 3. AIMS authoring environment

decomposition of the authoring process into a sequence of phases and decomposition of the authoring process in main activity modules (related to the educational system components). The authoring activities are independent of the system's domain, the educational strategy and the educational goal. The authoring task ontology describes the relations among the authoring tasks and the roles of domain objects, which they play in a particular task [5]. Each authoring task is defined by: (1) a sequence of activities with their activity type, constraints and input/output resources, with their resource type and constraints; (2) a goal; (3) requirements; (4) constraints, which can be flexible or hard constraints. Following this we define a set of:

- generic nouns reflecting the roles of the objects in the authoring process (e.g. concept, learning activity, structure, resource, course lesson, domain, author, student, text, relationship, goal, constraints, etc.)
- generic verbs representing authoring activities over the objects and applied in a combination or sequence of activities with specific objects or concepts and their modifications (e.g. modify, edit, assign return, update, select, check, etc.)
- generic adjectives representing the modifications of the objects and applied for modification and identification of these objects' attributes (e.g. shared, finished, required, idle, in-use, updated, etc.)
- other authoring task specific concepts (e.g. concent, prerequisite, lesson constraint, constraint satisfaction, state, attribute, predicate, knowledge, etc.)

The primitive functions are defined on objects (e.g. concepts, documents, course topics and tasks, and so on) within a specific structure such as course structure, goal

hierarchy, teaching action sequence, etc. They express a simple functional formalism, where the object changes the structure, or the structure is manipulated. Examples of atomic authoring functions include:

*CREATE (Structure)*, *CREATE (Object, Structure)*, *ADD (Object, Structure)*, *DELETE (Structure)*, *DELETE (Object, Structure)*, *VIEW (Objects)*, *EDIT (Object, Structure)*, *LIST (Objects, Structure)*, *UPDATE (Object, Structure)*, *UPDATE (Structure)*, where *Object*  $\in$  *Domain\_Concepts*, *Course\_Topics*, *Course\_Tasks*, *Resources*, and *Structure*  $\in$  *Domain\_Model*, *Course\_Model*, *Resource\_Base*.

The hierarchy of higher-level tasks represents conceptual categories of relationships (interdependence) between primitive functions. These present certain aggregation criteria (including causal and other relations among components) that are used for grouping primitive functional concepts into higher-level authoring functions (classes). This way we can construct/identify functional groups of authoring tasks. The higher-level functions represent a role of one base function for another base function. They are concerned not with the actual change in the objects, but with their actual function in the process of authoring. Examples of links realizing it include: *is-a-prerequisite-for*, *is-assigned-to*, *is-achieved-by*, *follows-from*, *is-preceded-by*, *requires*, *is-followed-by*, *if-<goal>-then-<action>*. (Note that the authoring tool shown in Fig. 1 already provides a means for creating such relationships in the limited context of the AHA! system.) Examples of some higher-level authoring tasks include: *DeleteLinks*, *DeletePath*, *DeleteAll*, *LinkDocTask*, *LinkDocConcept*, *CopyCourse*, *CompareObjectsStructure*, *ExistObjectStructure*. All the authoring tasks are grouped in relation to domain model, user model resource management and instructional-task model construction, in order to build a hierarchical organization of concepts linked by the ontological link types *is-a*, *part-of* and *attribute*. With the above description we have formed a functional basis for the authoring support in adaptive educational systems and have illustrated some of the benefits of an ontology-based approach for an authoring framework. In the next section we present extension of the AHAM reference model in order to achieve author-friendly primitives in terms of which the author can easily describe the goals and constraints of their instruction.

### 4.3 The LAOS Authoring Model

LAOS [17] is a general model for adaptive hypermedia authoring that tries to respond to the problems listed above. It is built to express this multitude of alternatives in an easy-to-edit manner, allowing *collaborative* (many authors working together on an adaptive hypermedia design task) or *incremental* authoring (same author working at different moments in time on the same adaptive hypermedia design task, e.g. after having some first students' feedback, or introducing new requirements from the head office, etc.). LAOS also allows automatic authoring techniques. It is based on the AHAM model [31], which in turn extends the Dexter reference model [20] for the specific field of adaptive hypermedia.

The first difference between LAOS and AHAM is that LAOS has an extra layer: the *goals and constraints model*. AHAM is best suited for applications in which

the user needs to explore the whole information space. Specifying goals enables the system to give a more focused presentation. Through constraints the search space can be reduced. Figure 4 shows LAOS as an extension of AHAM [31]. It consists of five layers: the *domain model*, the *goal and constraints model*, the *user model*, the *adaptation model*, and the *presentation model*.

The other difference between AHAM and LAOS is contained in the granularity and exact composition of elements in the top layers, as well as in the introduced operators. These are a result of the aim of adding automatic authoring techniques, which imply the authoring result to contain the necessary information for the system to process in an expressive manner, in other words, to be based on a good ontology. Therefore we looked for a better expression for both the static and the dynamic components of the AHS. In other words, we considered where the best place (in which layer) would be to store certain information and how. For example, concept relations should be in the domain layer, as in previous adaptive hypermedia systems, just as long as they express a real conceptual link between two or more concepts (content-driven relations). However, relations like the *prerequisite relationship type* described previously are related to the pedagogical intentions of the author (teacher) and not to the actual content of the concept. Therefore, such a relation should be in a different layer (in LAOS, on the goal and constraints layer).

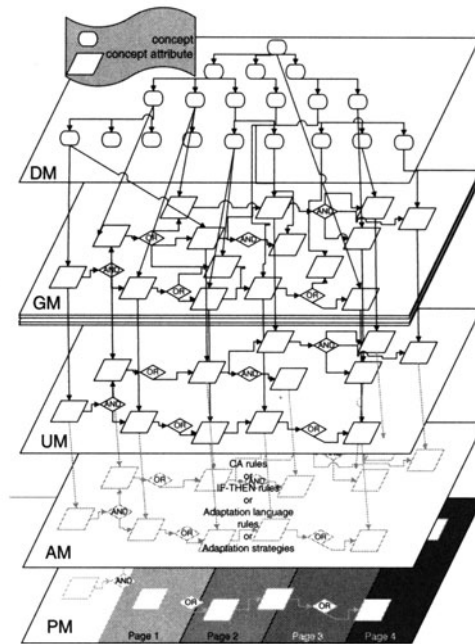
Another example of separation of concerns is the presentation. AHA! and also AHAM mix presentation-related issues, such as link colors, for instance, with concept content-related issues, such as concept relations. The presentation-related issues are very often client dependent and should be kept separately, so they can be dealt with directly. Therefore the need of the presentation layer (model).

The authoring steps of adaptive hypermedia with the LAOS model are presented in [17], where also a more detailed description about the components and functionality of each layer and the respective operators that implement this functionality can be found. An important feature of LAOS is that it was developed to work with a new adaptation model, LAG [15], that allows views over the different levels of adaptation, responding to the problem of authoring complexity from the capital adaptive hypermedia feature: the adaptation feature.

#### 4.4 MOT: My Online Teacher

LAOS and LAG are more general models that are applicable for general adaptive hypermedia analysis and authoring. The tool My Online Teacher (MOT) [16], however, was designed specifically for distance learning. Its predecessor is MyET (My English Teacher) [18], developed at the University of Electro-Communications, Tokyo, Japan. MOT is gradually implementing the ideas and definitions introduced by LAOS and LAG.

MOT is a hypermedia tool, developed and extended at the Eindhoven University of Technology, that can be used for authoring adaptive hypermedia courses. With this tool, the subject matter of the course to be designed can be modeled by means of concept maps (Fig. 5). Based on these concept maps lessons can be constructed. Concept maps and lessons form the two levels of preadaptive content, and they are



**Fig. 4.** LAOS: the five-layer authoring model of adaptive hypermedia

stored in a database. This structure lays the basis for various types of adaptation, as it uses both the expressivity of metadata annotation and the flexibility of the database structure (on which different queries can be performed). MOT is being extended so it can interface with AHA!. This will then allow us to implement LAOS and LAG flexibility in the AHA! system.

MOT can demonstrate the ideas of separating the domain model and the goal and constraints model, as well as some of the ideas on automatic authoring (these include automatic generation of links within the concept domain model and automatic generation of an instance of the goal and constraints model from an instance of the domain model).

## 5 Concluding Remarks and Outlook

Adaptive educational hypermedia applications have the potential to give online course material a 'personal touch'. Several successful applications and application frameworks exist, but mass employment of adaptive hypermedia in education is still lacking. We believe that authoring difficulties are the main problem that remains. We have shown new approaches to make authoring easier, and to allow adaptation to other aspects besides the system's idea of the user's knowledge of concepts.

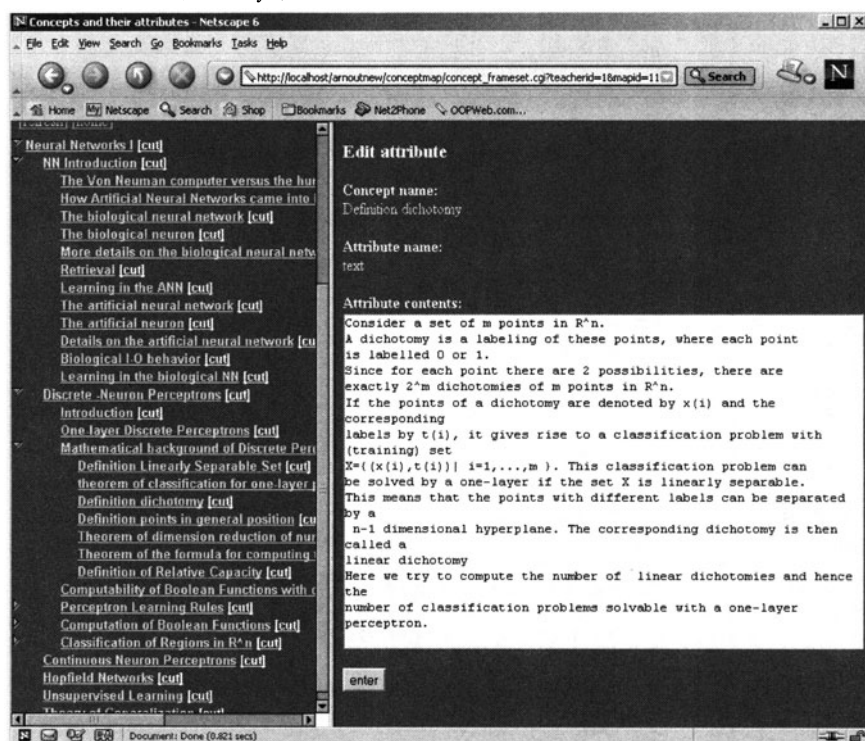


Fig. 5. MOT: My Online Teacher

A second issue that needs to be tackled in order to make adaptive educational hypermedia popular is the current 'closed' nature of the applications that exist. Many infrastructures exist to integrate learning material from different sources into large information sources. Yet these infrastructures lack the ability to handle adaptive sources of learning material. The definition of standards in the area of adaptive educational hypermedia, in collaboration with the ongoing learning technology standards developments, is needed to enable the exchange of course material and user model information between adaptive applications.

## Acknowledgements

Work on AHA! 1.0 was supported by a grant of the Eindhoven University of Technology. The development of AHA! 2.0 and 3.0 is supported by a grant of the NLnet Foundation (<http://www.nlnet.nl/>). Work on LAOS, LAG and MOT is linked to the European Community Socrates Minerva project 'Adaptivity and adaptability in ODL based on ICT' (project reference number 101144-CP-1-2002-NL-MINERVA-MPP). Work on AIMS and ATO is supported by the University of Twente and MizLab, at Osaka University.

## References

1. L. Aroyo. *Task-based approach to information handling support with Web-based education*. PhD thesis, University of Twente, 2001.
2. L. Aroyo and D. Dicheva. Aims: Learning and teaching support for WWW-based education. *International Journal for Continuing Engineering Education and Life-long Learning*, 11(1/2):152–164, 2001.
3. L. Aroyo, D. Dicheva, and A. Cristea. Ontological support for Web courseware authoring. In *Proc. 6th International Conference on Intelligent Tutoring Systems*, volume LNCS 2363, pages 270–280. Springer, Berlin Heidelberg New York, 2002.
4. L. Aroyo, D. Dicheva, and I. Velez. A concept-based approach to support learning in a Web-based course environment. *AI in Education: Frontiers of AI and Applications*, 68:1–10, 2001.
5. L. Aroyo and R. Mizoguchi. Authoring support framework for intelligent educational systems. 2003.
6. P. De Bra, A. Aerts, D. Smits, and N. Stash. AHA! version 2.0, More adaptation flexibility for author. In *Proc. AACE ELearn Conference*, pages 240–246, 2002.
7. P. De Bra and L. Calvi. Creating adaptive hyperdocuments for and on the Web. In *Proc. AACE WebNet Conference*, pages 149–154, 1997.
8. P. De Bra and L. Calvi. AHA! an open adaptive hypermedia architecture. *The New Review of Hypermedia and Multimedia*, 4:115–139, 1998.
9. P. De Bra, G.J. Houben, and H. Wu. AHAM: A Dexter-based reference model for adaptive hypermedia. In *Proc. ACM Conference on Hypertext and Hypermedia*, pages 147–156, 1999.
10. P. Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6:87–129, 1996.
11. P. Brusilovsky. Adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 11:87–110, 2001.
12. P. Brusilovsky, J. Eklund, and E. Schwarz. Web-based education for all: A tool for developing adaptive courseware. *Computer Networks and ISDN Systems (Proc. International World Wide Web Conference)*, 30(1-7):291–300, 1998.
13. L. Calvi and P. De Bra. Using dynamic hypertext to create multi-purpose textbooks. In *Proc. AACE ED-MEDIA Conference*, pages 130–135, 1997.
14. A. Cini and J. Valdeni de Lima. Adaptivity conditions evaluation for the user of hypermedia presentations built with AHA! In *Proc. International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 497–500, 2002.
15. A. Cristea and L. Calvi. The three layers of adaptation granularity. In *Proc. ninth International Conference on User Modelling (UM'03)*, pages 4–14. Springer, Berlin Heidelberg New York, 2003.
16. A. Cristea and A. De Mooij. Adaptive course authoring: MOT, my online teacher. In *Proc. of ICT-2003, IEEE LITF International Conference on Telecommunications, "Telecommunications + Education" Workshop (Feb 23 - March 1)*, pages CD-ROM, 2003.
17. A. Cristea and A. De Mooij. LAOS : Layered WWW AHS authoring model and their corresponding algebraic operators. In *Proc. Twelfth International World Wide Web Conference (WWW'03)*, 2003.
18. A. Cristea, T. Okamoto, and S. Belkada. Concept mapping for subject linking in a WWW authoring tool: MyEnglishTeacher: Teachers' site. In *Proc. ANNIE'00, Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming and Complex Systems*. ASME Press, 2000.



19. D.P. da Silva. Concepts and documents for adaptive hypermedia: A model and a prototype. In *Proc. Workshop on Adaptive Hypertext and Hypermedia*, pages 33–40, 1998.
20. F. Halasz and M. Schwartz. The Dexter hypertext reference model. *Communications of the ACM*, Vol. 37, nr. 2, pages 30–39, 1994.
21. J. Kay and B. Kummerfeld. An individualised course for the C programming language. In *Proc. Second International World Wide Web Conference*, 1994.
22. Y. Kitamura, T. Sano, and R. Mizoguchi. Functional understanding based on an ontology of functional concepts. In *Proc. of PRICAI'00 Conference*, pages 723–733, 2000.
23. N. Koch. *Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process*. PhD thesis, Ludwig-Maximilians-Universität München, 2001.
24. N. Koch and M. Wirsing. The Munich reference model for adaptive hypermedia applications. In *Proc. Second International Conference on Adaptive Hypermedia and Adaptive Web-based Systems*, pages 213–222, 2002.
25. A. Mitrovic. Experiences in implementing constraint-based modeling in SQL-Tutor. In *Proc. 4th International Conference on Intelligent Tutoring Systems*, pages 414–423, 1998.
26. A. Mitrovic. Using evaluation to shape ITS design: Results and experiences with SQL-Tutor. *User Modeling and User-Adapted Interaction*, 12(2-3):243–279, 2002.
27. R. Mizoguchi, K. Sinitsa, and M. Ikeda. Task ontology design for intelligent educational/-training systems. In *Proc. 3rd International Conference on Intelligent Tutoring Systems*, 1996.
28. C. Romero, S. Ventura, P. De Bra, and C. de Castro. Discovering prediction rules in AHA! courses. In *Proc. International Conference on User Modeling*, pages 25–34, 2003.
29. G. Weber and P. Brusilovsky. ELM-ART: An adaptive versatile system for Web-based instruction. *International Journal of Artificial Intelligence in Education*, 12:351–384, 2001.
30. G. Weber and M. Specht. User modeling and adaptive navigation support in WWW-based tutoring systems. In *Proc. International Conference on User Modeling*, pages 289–300, 1997.
31. H. Wu. *A Reference Architecture for Adaptive Hypermedia Applications*. PhD thesis, Eindhoven University of Technology, 2002.

---

# MP<sup>3</sup> – Mobile Portals, Profiles and Personalization

Barry Smyth<sup>1,2</sup> and Paul Cotter<sup>2</sup>

<sup>1</sup> Smart Media Institute, University College Dublin, Dublin 4, Ireland

barry.smyth@ucd.ie

<sup>2</sup> ChangingWorlds Ltd., Trintech Building, South County Business Park,  
Leopardstown, Dublin 18, Ireland

paul.cotter@changingworlds.com

**Summary.** The mobile Internet promises a new era of access to information anytime, anywhere. This has been made possible by a variety of technology and business developments, with mobile operators and device manufacturers at center-stage. However, to date the reality of the mobile Internet has not lived up to the promises made, and consumer take-up has been slow. This is especially true in the context of mobile (WAP – wireless application protocol) portals, which attempt to replicate the success of traditional Internet portals on Internet-enabled mobile handsets and personal digital assistants (PDAs). In this article we examine mobile portals in detail and argue that the current problems are symptomatic of serious usability issues that make it difficult for users to efficiently access available information and services. In turn, we describe how recent advances in so-called ‘personalization technologies’ may hold the key for mobile operators by allowing their portals to automatically adapt to the needs of individual users.

## 1 Introduction

Recent advances in wireless access and display technologies, circuit design, embedded software systems and communications protocols have led to the emergence of a wide range of compact but powerful computing terminals, such as personal digital assistants (PDAs) and WAP-enabled mobile phones, which in turn have fueled the development of the mobile Internet. The promise of the mobile Internet is a compelling one: instant access to relevant information and a range of online services, from e-mail and news to games and maps, all from your mobile phone, PDA or laptop. However, the convergence of wireless and Internet technologies has been a slow and complicated process, involving many different stakeholders (mobile operators, device manufacturers, infrastructure vendors, content producers and aggregators, advertisers, end users, etc.), and although the marketing drive has led to high consumer expectations many practical issues remain to be resolved.

Mobile portals are a vital part of the mobile Internet vision. They represent an attempt to reproduce the success of portal services on the Internet, but through mobile handsets and PDAs. In this article, while we provide a general backdrop to the mobile

Internet, we focus on mobile portals and the problems that have resulted in a less than enthusiastic response from consumers. In general, mobile portals, as they exist today, have largely failed to live up to user expectations. The reasons for this are many and varied, but a key problem relates to usability [3, 8, 34, 41, 42]. In the past the usability of mobile portals has been compromised, mainly due to poor portal design, but exacerbated by limited device functionality, bandwidth and content. Fortunately, the new generation of mobile services (so-called 2.5G and 3G services) represents a significant improvement on this state of affairs; the bandwidth and content issues have largely been, or are being, resolved, and the latest phones offer users significant interface and functionality improvements over the original models. However, poor portal design remains an issue. The core problem is that the menu-driven nature of mobile portals, whereby users access content services by navigating through a series of hierarchical menus (Fig. 1), means that users are spending a significant amount of their time online navigating to content (their *navigation time*) and limited time interacting with content (their *content time*).

This distinction between navigation and content time is especially clear in the current generation of mobile portals due to their structure, which clearly separates menu pages from content pages, and it is important from a usability perspective. For the end user, content is king, and navigation is a means to an end, a necessary evil in the quest for compelling content. An ideal portal should present a user with relevant content without the need for spurious navigation. This ideal is not achieved by today's mobile portals, and this frustrates users and limits the efficiency of mobile information access. Moreover, by charging users for their navigation time or accesses (as well as their content time and accesses) mobile operators are adding insult to injury.

In this article we describe how automatic user modelling and personalization techniques can be used to help solve this navigation problem by automatically adapting the structure of a mobile portal for the needs of individual users. In the following sections we briefly review the current state of the mobile Internet, focusing in particular on mobile portals and their usability. We describe how navigation effort can be modelled as *click-distance* – the number of menu *selections* and *scrolls* needed to locate a content item – and we demonstrate how personalized navigation techniques can reduce click-distance and thus radically reduce navigation effort. These reductions translate into significant usability improvements, which we demonstrate through a comprehensive live-user evaluation. For example, we show that for every one second of navigation effort saved, the average user is willing to engage in additional three seconds of content time. Finally, we conclude with a brief outline of a number of important open issues that remain in this research field.

## 2 The Mobile Internet

The mobile Internet is made up of a group of related infrastructure, protocol and device technologies, allowing the end user to access various types of data services from their mobile devices. These services typically include Web-style information



**Fig. 1.** A selection of menu pages from a modern mobile (WAP) portal. The example illustrates a sequence of menus leading to a cinema listings service. The user is expected to navigate to the *Ormonde* cinema listings page by scrolling to, and selecting, the appropriate intermediate menus (*Entertainment*, *Movie Times*, *Dublin Cinemas*). A total of 15 clicks are required, made up of 4 menu selections and 11 menu scrolls

content, e-mail services, games, etc. And they can be accessed using a range of devices from limited, first-generation WAP (Wireless Application Protocol) phones to today's sophisticated PDAs and so-called smart phones. In recent times the mobile Internet has been the subject of intense research [1, 2, 4, 10, 20, 27, 29, 33, 36, 46]. Although this article focuses on a specific topic within the mobile Internet space, the usability of mobile portals, this section provides a more general backdrop to the mobile Internet and to our own research.

## 2.1 The Evolution of the Mobile Internet

Mobile services are usually described as either first-, second- or third-generation network technologies, with second-generation networks the norm today. Third-

generation (3G) networks have enjoyed considerable publicity in recent times because of their promised ability to move data at high speeds and the potential applications that such capabilities apparently support (e.g., video-conferencing). 3G networks are a progression that began with the advent of first-generation (1G) analog technologies, which carried the original mobile telephony voice services. Today most carriers have either upgraded, or are in the process of upgrading, from 2G to 2.5G networks, with a eye on 3G in the near future.

The original mobile data services, supported by 2G networks, were characterised by limited-bandwidth (9.6/14.4 kbps). Moreover, being circuit-switched, users experienced considerable delays connecting to the mobile Internet. For the most part, in Europe at least, the mobile Internet has used the WAP standard to deliver content services as WML (Wireless Markup Language) pages; see <http://www.wapforum.org/> for further information on WAP and related technologies. WAP is a communications and applications protocol designed for the delivery of mobile data services across most wireless networks. WML is a mark-up language based on XML and is similar in nature to HTML; it is used to create pages (largely text-based) that can be displayed in any standard WAP browser on a WAP-enabled mobile phone.

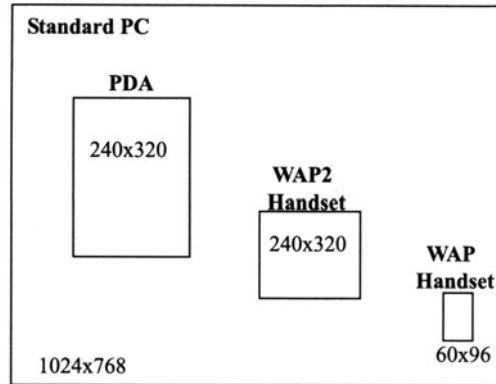
Recent upgrades to 2.5G services, based on the General Packet Radio Service (GPRS) standard, offer improved bandwidth of up to 128 kbps and, perhaps more importantly, always-on connectivity. In other words, the connection delay experienced by early mobile Internet users with 2G services has been largely eliminated, allowing for improved data access and service delivery. In parallel with this infrastructural change, a new generation of more sophisticated handsets has become available to offer users an enhanced mobile Internet experience, with greater graphical impact.

Looking to the future, 3G technologies will further increase the available bandwidth (up to 2 Mbps) to allow mobile devices to offer high-speed Internet access, plus video and CD-quality music services. So far, at the time of writing (Spring 2004) the commercial rollout of 3G services has been limited to Japan (since 2001) and the UK (since 2003).

## 2.2 Mobile Internet Devices

From a user experience viewpoint, one of the key features of the mobile Internet is the degree to which existing consumer devices (such as WAP phones and PDAs) represent a significant step backwards in terms of their functionality, at least when compared to the traditional Internet device (the desktop PC or laptop). In particular, presentation and input capabilities tend to be extremely limited on most mobile devices (see [7, 8, 45, 13, 18, 21, 24, 40]).

For example, Fig. 2 illustrates the relative differences in the screen sizes for a range of common devices. Consider, for instance, a typical desktop PC with a screen size of  $1024 \times 768$  pixels. Modern PDAs have screen resolutions of only  $240 \times 320$  pixels, less than one tenth of the screen real-estate of the typical PC. This situation is considerably worse for a typical WAP phone, with screen sizes of  $60 \times 96$  pixels being commonplace, more than 100 times smaller than the PC's screen. Even the latest mobile handsets, which boast large screens, offer only  $176 \times 208$  pixels, which is still



**Fig. 2.** Relative screen sizes (pixels) for traditional PC, PDA and mobile handset screens

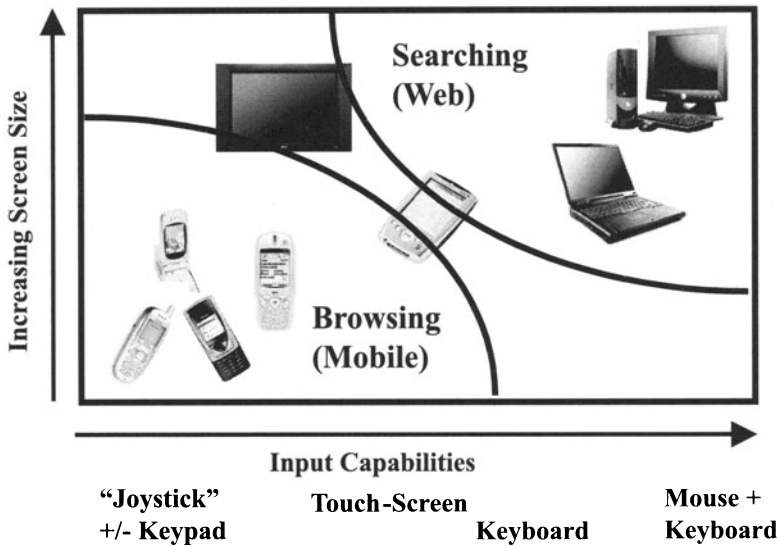
more than 20 times smaller than the PC screen. The bottom line is obviously that the ability to present large amounts of information on a mobile handset is significantly compromised in comparison to a traditional PC. As such, the importance of presenting relevant information becomes all the more significant.

Mobile handsets are also limited by their capacity to receive user input. The keyboard and mouse functionality of a modern PC are notably absent, and the mobile phone numeric keypad makes it extremely difficult for users to input any quantity of information. From a mobile portal viewpoint, these devices restrict selection features to simple ‘scroll’ and ‘select’ actions that allow the user to scroll through menu lists and indicate simple option choices. Some improvements are present in most PDAs, which tend to offer touch-sensitive screens that are easier to manipulate. Nevertheless, data input remains difficult at best, and the indications are that previous predictions regarding the convergence of mobile phone and PDA devices are unlikely [16], leaving mobile phones in particular with a severely limited input capability.

### 2.3 Mobile Information Access

The differences in device characteristics, in terms of screen-size and input capabilities, that exist between mobile handsets and more traditional Internet devices, such as PCs and laptops, directly influence the manner in which users access information using these devices (Fig. 3). For example, on the Internet today search has largely become the primary mode of information access. It is relatively easy for users to input search queries and search engines have improved significantly in their ability to respond intelligently to user needs. In addition, the large screen sizes make it feasible for users to efficiently parse the long lists of search results returned.

In contrast, search is far more problematic on mobile devices. Entering queries is simply too time consuming and complex for the average user to tolerate, and small screen sizes make it practically impossible for users to process the result lists returned. As a result, browsing is the primary mode of information access on the mobile Internet. Instead of searching for information, users attempt to navigate to information by using



**Fig. 3.** Because of their small screen sizes and limited input capabilities, browsing is the primary mode of information access on mobile devices. This is in contrast to more conventional Internet devices, such as desktop or laptop PCs, where searching is the more usual mode of information access

mobile portals. Today the vast majority of mobile Internet services are accessed via an operator portal, with direct search constituting a small fraction ( $< 10\%$ ) of activity.

This distinction between alternative modes of information access on the mobile and fixed Internet is an important one that frames our own research. We suggest that to help users to locate information and services more effectively on the mobile Internet, we must attempt to improve the efficiency of mobile portal browsing or navigation (see also [13]). Of course, this remains to be proven and research in this area is fragmented at best. Sellen et al. [37, 38] look to the future of the mobile Internet in general and ask the question: To what extent should we rely on familiar notions of Web browsing when we think of the Internet in a mobile context? In an attempt to answer this question Sellen and Murphy perform a task analysis of traditional desktop browsing activities, distinguishing between six different activity types: ‘transacting’, ‘browsing’, ‘communicating’, ‘finding’, ‘housekeeping’ and ‘information gathering’. They conclude that many of the ‘finding’ and ‘browsing’ activities that are commonplace on the Web may also be suitable for the mobile Internet, but that other activities are less suitable. For example, they highlight that many of the ‘finding’ activities are well suited to a small screen because of the limited type of information being sought (e.g., train times, phone numbers or directions). We would add a cautionary note here. Screen size is not the only factor that must be considered when evaluating application suitability. Mobile devices have very limited input capabilities too, and we would argue that this seriously compromises the suitability of certain information-finding

activities because it makes it difficult for the user to specify even the simplest query. Sellen et al. do not appear to give due consideration to the impact of limited input features in their task analysis. Certainly, in our experience, the dominant form of information access through mobile portals is based on browsing activity, with query-based search falling far behind. For this reason improvements to browsing activity are likely to pay handsome dividends in the short term.

### 3 Mobile Portal Usability and Click-Distance

So far, mobile portals have failed to live up to market and user expectations. Important factors, such as those highlighted above in relation to infrastructure and device functionality, have played a critical role in this state of affairs. In addition, only limited high-quality content was available from the early mobile portals. While recent infrastructure improvements (specifically the upgrading of networks to 2.5G) and a more intelligent investment in mobile content has resulted in significant improvements, overall usability remains a key problem, limiting the ability of users to easily locate, and benefit from, wireless content. Indeed this usability problem represents a key ongoing research issue as reflected in a growing body of recent research efforts (for example, [5, 8, 9, 45, 11, 16, 17, 18, 21, 24, 25, 28, 32, 40, 41, 42, 44, 48]).

#### 3.1 The Usability of Hierarchical Menu Systems

Perhaps the core usability problem with mobile portals is related to navigation: users spend significant time laboriously navigating to content through a series of menus. Mobile portals are examples of hierarchical menu systems (HMS) [25], and long before the arrival of the mobile Internet different forms of hierarchical menu systems were studied extensively with respect to their general usability and navigation characteristics [15, 19, 22, 23, 26, 43, 47, 49, 50]. Very briefly, much of this previous research focused on the structural properties of hierarchical menu systems, for example, their depth and width, as they relate to the ability of a user to easily navigate through the HMS. For instance, Miller [26] discovered that for moderate sized menu systems, wide hierarchies are preferable to deep hierarchies due to the short-term memory limitations of end users, which led to a greater number of navigation errors in deep hierarchies; the interested reader is also referred to [19, 43] for related work. Compatible observations have been made with respect to the menu hierarchies found in the World Wide Web. For example, Zaphirs [50] performed a related study for the hyperlink menu structures of Web pages and reported similar findings in relation to depth. Jacko and Salvendy [15] reported similar error-rate results associated with deep hierarchies and also highlighted the relationship between navigation time, perceived menu complexity and hierarchy depth; specifically, test subjects perceived deep hierarchies to be more complex than wide hierarchies of a similar size.

So the evidence is clear that the complexity of a hierarchical menu system has a significant impact on its usability and the ability of users to navigate through menu levels. The type of menu hierarchies found in mobile portals are likely to be subject to similar findings [8, 25].



### 3.2 The Click-Distance Model of Navigation Effort

The scale of the usability and navigation problems associated with mobile portals today, and the mismatch between user expectations and realities, is highlighted by a number of recent studies [45, 11, 34]. For instance, Chittaro and Dal Cin [45] examine two important WAP user-interface design choices – single-choice menu selections and navigation among cards – with respect to novice users. They provide evidence that exploiting such navigation links and single-choice selections can significantly improve usability. However, another study highlights the pitfalls of too many navigation links and claims that while the average user expects to be able to access content within 30 seconds, the reality is closer to 150 seconds [34].

Clearly, the time that it takes a user to locate and access a specific content item is a measure of navigation effort. However, measuring navigation effort by timing studies is laborious and expensive. In our own research we emphasised the importance of developing a predictive model of navigation effort that avoids the need for laborious timing studies [41, 42]. To this end we have developed the so-called *click-distance* model of navigation effort. This model is based on the assumption that the navigation effort associated with an item of content depends critically on the location of that item within the portal structure, and, specifically, on the number of ‘navigation steps’ that are required in order to locate and access this item from a given starting position within the portal (typically the portal home page).

What is a ‘navigation step’? With the current generation of mobile phones, there are two basic types of navigation step. The first is the *menu select*: the user clicks to select a specific menu option. The second is a *menu scroll*: the user clicks to scroll up or down through a series of options. Accordingly, an item of content  $i$  within a mobile portal, can be uniquely positioned by the sequence of selects and scrolls needed to access it, and the navigation effort associated with this item can be simply modelled as click-distance, the number of these selects and scrolls (Eq. (1)).

$$ClickDistance(i) = Selects(i) + Scrolls(i) \quad (1)$$

Although this simple model of navigation effort equally weights the scrolls and selects, when we evaluate click-distance in comparison to navigation time, by analyzing the behaviour of live users on commercial mobile portals, we find a near-perfect correlation. For instance, the results of a recent evaluation (based on 6 weeks of WAP usage for over 100 users, Sect. 5) indicate a correlation coefficient of 0.92 between click-distance and navigation time. Thus, the click-distance of a content item is a strong predictor of the navigation time associated with its access; see also related work by [8, 25].

Crucially, large click-distances are indicative of protracted navigation times, and recent studies illustrate the extent of the click-distance problem. For example, an analysis of 20 European mobile portals reported an average click-distance in excess of 16 [39]. In other words, a typical European mobile portal user can expect to have to make 16 or more clicks (scrolls and selects) to navigate from their portal home page to a typical content target. Moreover, on average European portals are organised such

that less than 30% of content sites are within 10–12 clicks of the portal home page; 10–12 clicks corresponds to a navigation time of about 30 seconds [41, 42], which is expected by mobile portal users [34]. To put this another way, more than 70% of mobile portal content is essentially invisible to users because of its positioning within its parent portal.

## 4 Personalized Navigation

We have argued so far that the fundamental problem with mobile portals is that their users are expected to spend too much time navigating to content, and that this greatly limits the value of their online experience on a number of levels. For example, one of the compelling arguments in favour of mobile portals is their promise of ‘information on the move’ – the idea that a user can quickly ‘dip’ in to their mobile portal while waiting for a bus or standing in line – which is very much contradicted by the inherent information access challenges.

The argument has been made that the current information access problems are likely to disappear as new technology and infrastructure developments are introduced. For example, 3G infrastructure promises much higher bandwidth and so offers more rapid downloads. Similarly, improvements in display technologies are likely to increase screen-sizes. However, while such developments are very much welcomed, and will no doubt have a positive impact on mobile portals, they are unlikely to solve the information access problem. For example, increases in bandwidth will have no impact on portal click-distance – increases in bandwidth may speed up the download of intermediate menu pages en route to content, but the need for manual user navigation will remain and will continue to represent a crucial bottleneck. Similarly, while large screens may allow for more options to be presented to the user, and thus reduce the number of selects needed, these selects will be replaced by the need for more scrolling. At any rate, screen-sizes are likely to remain limited due to the need to constrain the form factor of mobile devices in order to preserve their portability. The average consumer appears to be quite content to carry a small mobile phone, but they are less eager to carry PDA-style devices [16].

### 4.1 ‘One-Size-Fits-All’ Versus Personalization

We argue that large click-distances are a fundamental feature of a *one-size-fits-all* approach to portal design and that the only sustainable solution to the usability problem is to break with this tradition. Ultimately, portal click-distance can be greatly minimised by tailoring the portal for the needs of an individual user so that content and services that are of interest to this user are near to the portal home page, and thus accessible with a minimum number of clicks. Less relevant content and services should be relegated to the outskirts of the portal. Achieving this is no trivial task. It means developing a separate portal for each individual user, an unacceptably expensive task for the portal operator. Of course, instead of forcing the operator to be responsible for ‘individualizing’ the portal, the user could be provided with a facility that allows

for manual customization. However, to date such initiatives, whereby the user can manually reconfigure the portal according to their needs, have failed to attract users in sufficient numbers to be successful, and even those users who do initially spend time customizing their portal rarely maintain it in line with their changing interests.

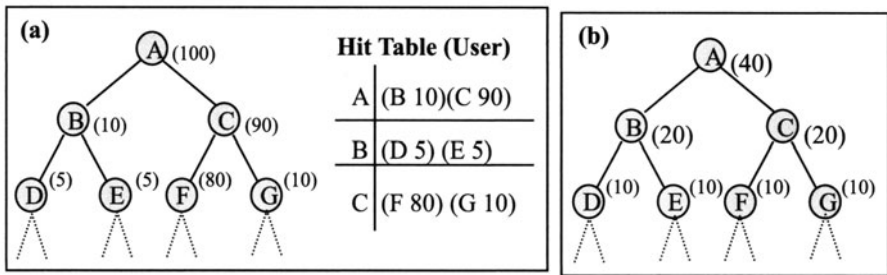
However, a solution is at hand that avoids the need for manual customization or major operator expense. Recent research has made it possible to use user profiling and personalization techniques to learn about the preferences of individual users in order to strategically adapt the structure of the portal on a user-by-user basis. For example, if a given user regularly accesses her local cinema's listings at the weekend, then this content service can be made available from the portal home page (or at least nearby to the home page) at these times, rather than languishing deep with the portal structure. Thus, our strategy for decreasing navigation effort is to reduce the click-distance of the content items that a given user is likely to be interested in by *promoting* these items (or the links that lead to them) to higher positions within the portal menu structure. In general, personalization research seeks to develop techniques for learning and exploiting user preferences to deliver the right content to the right user at the right time [6, 12, 30, 31, 35, 40], and these ideas can be applied to the personalization of a portal structure to aid navigation effort [3, 41, 42].

The basic idea behind our personalized navigation technique is to use a probabilistic model of user navigation preferences to predict the likelihood that some menu option  $o$  will be selected by a user  $u$  given that they are currently in menu  $m$ , and based on their past navigation history. We wish to compute  $P_u(o|m)$ , the access probability of  $o$  given  $m$  for user  $u$ , for all options  $o$  accessible from  $m$  (either directly or indirectly, through descendant menus). Put simply, when a user arrives at menu page  $m$ , we do not necessarily return the default options,  $o_1, \dots, o_n$ . Instead we compute the options,  $o'_1, \dots, o'_k$ , that are most likely to be accessed by the user from  $m$ ; that is, the  $k$  menu options accessible from  $m$  that have the highest access probabilities. This can mean promoting certain menu options up to  $m$ , menu options that by default belong to descendant menus of  $m$ . The size of the final personalized menu is constrained by some maximum number of options  $k$ , and the constituent options of  $m$  are ordered according to their access probabilities.

## 4.2 Navigation Profiles

As users access a portal over time they build up a navigation history. Each time a user accesses a particular item of content they do so by navigating through a series of portal menus and options. Frequent accesses to the same content and services lead to well-travelled paths through the portal, and by recording these access patterns – that is, by recording each sequence of menu options that are accessed – it is possible to construct an accurate picture of an individual user's navigation history [14]. The so called *hit table* data structure is an efficient way of storing this information for a given user; see Fig. 4 for an example of a partial menu tree and corresponding hit table entries. A hit table is basically a hash-table keyed according to the menu ID and storing the number of accesses made by that user to options within that particular

menu. For example, Fig. 4a indicates that the particular user in question has accessed option *B* of menu *A* 10 times, and option *C* 90 times.



**Fig. 4.** Menu trees and hit tables: (a) A partial menu tree corresponding to a user hit table. (b) A static menu tree corresponding to a static hit table

The critical thing to understand about the hit table is that it provides a means by which to reconstruct a portal structure that reflects a given user's past access patterns. The hit table entries can be used directly to compute the probabilities that a given menu option will be accessed within the portal, as we will demonstrate in the next section. But first it is worth highlighting the need for two hit tables: a global static hit table that is initialized with respect to the default portal structure (Fig. 4b); and a user hit table that records each user's particular history on the portal (Fig. 4a). The static table makes it possible to deliver the standard (default) menu structure (as developed by the portal designer) early on, but this will eventually be over-ridden by the personalized menu once the access probabilities build. Moreover, the default hit values that are set in the static hit table make it possible to control the personalization *latency*: low values mean that personalization takes effect very quickly, while large values make the system less sensitive to user activity.

### 4.3 Building a Personalized Menu

The key then to personalizing the navigation structure of a mobile portal relies on the ability to reconstruct individual portal menus to reflect the navigation history of a given user. For example, if a user regularly navigates from the 'entertainment' menu of a portal through a series of submenus in order to access their local cinema listings, then perhaps this local cinema listings option should be listed as one of the options in the 'entertainment' menu, in addition to any default options placed there by the portal designer.

To reconstruct a personalized version of menu *m* we must decide which, if any, of the menu options that appear below *m* in the default portal structure should be added to *m*. In general, there could be many eligible options, but for practical reasons not all of them should be added to *m*. For example, menu size will necessarily be limited in mobile portals. Thus, a means of ordering eligible options is required. One solution is to compute the *k* most probable options from *m*; that is, the *k* options with the

highest  $P_u(o|m)$ . In other words, the  $k$  options that are most likely to be accessed, given that the user is currently accessing menu  $m$ , would be added to  $m$ . To do this we take account of the hit values listed for each option in both the static and user hit tables by using the recorded access frequencies as a way to estimate the necessary access probabilities. For the data shown in Fig. 4 the following access probabilities are determined:

$$\begin{aligned}
 P_u(B|A) &= (20 + 10)/(40 + 100) &= 0.214 \\
 P_u(C|A) &= (20 + 90)/(40 + 100) &= 0.786 \\
 P_u(D|A) = P_u(B|A)P(D|B) &= (30/140)(10 + 5)/(20 + 10) &= 0.107 \\
 P_u(E|A) = P_u(B|A)P(E|B) &= (30/140)(10 + 5)/(20 + 10) &= 0.107 \\
 P_u(F|A) = P_u(C|A)P(F|C) &= (110/140)(10 + 80/20 + 90) &= 0.642 \\
 P_u(G|A) = P_u(C|A)P(G|C) &= (110/140)(10 + 10)/(20 + 90) &= 0.142
 \end{aligned}$$

Thus in descending order of access probability (or desirability) we have  $C$ ,  $F$ ,  $B$ ,  $G$ ,  $D$  and  $E$ . And for  $k = 3$ ,  $C$ ,  $F$  and  $B$  are selected, in order, for addition to menu  $A$ .

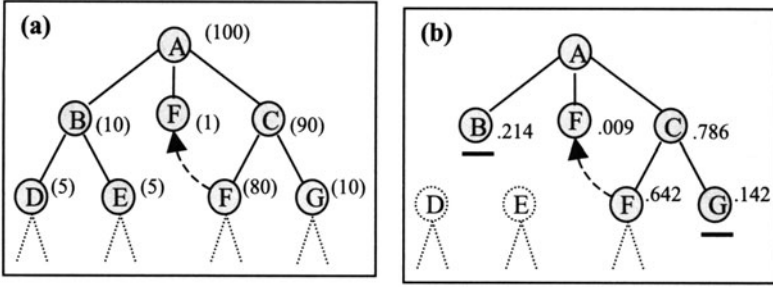
#### 4.4 Menu Promotions

This personalized navigation method supports two basic types of menu adaptation. First, a menu option may be *reordered* within the context of its default menu; that is, its relative position within its parent menu may be changed so that all menu options are ordered in descending order of their access probabilities. In this way, even if no new menu options are added to a menu, its default options can still be ordered according to the access probabilities so that those default options that are more likely to be accessed by the user are presented high up in the option list, ahead of options that are less likely to be accessed. This reordering directly influences click-distance by reducing the number of scrolls needed to locate a particular content item.

Alternatively, if there is sufficient evidence, a menu option may be *promoted* from its default menu to an ancestral menu. Thus promotion is the second type of menu adaptation and influences click-distance by reducing the number of menu selects needed to access a content item.

In this sense, menu reorderings and option promotions (and conversely demotions) are side effects of the access probability calculations and provide a fluid personalization scheme that gracefully adapts the navigation structure of a portal in response to a user's access patterns. For instance, in the above example option  $F$  is promoted into menu  $A$  from menu  $C$ ; see Fig. 5a. In theory, of course, options can be promoted from deeper levels of the portal structure once their probabilities build sufficiently, but in practice certain limits may be necessary to control the speed and scope of personalization. For example, users may become disorientated if menus are seen to change too rapidly or if options jump from low levels to high levels within a short time frame.

In the context of the example shown in Fig. 5, if  $F$  is subsequently selected from  $A$ , then it is added to  $A$ 's entry in the user's hit table. So the next time that menu  $A$



**Fig. 5.** (a) Option  $F$  is promoted into menu  $A$  from menu  $C$  and accessed once. (b) terminating search at  $B$  and  $G$  during the personalization of  $A$  for  $k = 2$

is created, and  $P_u(F|A)$  needs to be computed, we must account for the new hit data for  $F$  (see Fig. 5a). Specifically, assuming a single access to  $F$  as an option in  $A$ , we get:

$$P_u(F|A) = 1/101 + (110/141)(10 + 80/20 + 90) = 0.009 + 0.638 = 0.647$$

#### 4.5 An Efficient Algorithm for Computing Access Probabilities

Computational efficiency is a critical concern for any personalization technique. This is especially true for the above personalized navigation scheme given the need for real time personalization; that is, each menu  $m$  should be constructed in real time. Clearly, the efficiency of the proposed personalization method depends on the complexity of the process that identifies the  $k$  most probable options for the menu  $m$ . One issue in relation to this is that the reconstruction of a personalized version of menu  $m$ , as described, can mean examining not just the default options of  $m$ , but also all the options contained in menus that are descendants of  $m$ . This essentially means that a breadth-first search from  $m$  to the content leaves of the menu tree is required, and this may raise serious efficiency concerns in the case of large mobile portals, which contain hundreds or even thousands of menus and options.

Fortunately, a more efficient algorithm for reconstructing personalized menus is possible, one that does not require a full breadth-first search of the menu tree. The key to this algorithm is to recognize that, by definition,  $P_u(o|m)$  is always greater than, or equal to,  $P_u(o'|m)$ , where  $o'$  is an option of a menu  $m'$ , which is itself a descendent of  $m$  through  $o$ . This observation allows us to cut off the search for probable menu options at certain points in the menu tree. It means that we can find the  $k$  most probable options for menu  $m$  by performing a depth-limited, breadth-first search over the menu tree rooted at  $m$  because we only need to expand the search through an option  $o'$  if  $P_u(o'|m)$  is greater than the  $k$ th-best probability so far found.

For example, as Fig. 5b indicates, during the calculation of the access probabilities for  $A$ 's descendants with  $k = 2$ , search can be initially cut off at option  $B$ , since  $B$ 's children cannot have access probabilities greater than 0.214, which is the probability

of the  $k$ th-best option found so far ( $B$  itself). Similarly, after computing the access probabilities for  $C$ 's default options ( $F$  and  $G$ ), search can be cut off at  $G$ , since its probability is less than 0.642, the new  $k$ th-best option. In practice this technique can result in a significant reduction in search effort, allowing probabilities to be computed on-the-fly without a noticeable impact on performance.

## 5 Experimental Evaluation

At the start of this article we set our stall out in relation to mobile portal usability, arguing that protracted navigation times are an inherent problem associated with accessing content through a mobile portal, and that this navigation problem is leading to frustrated users and a poor overall user experience. In turn, we argued that personalization techniques offer a sustainable long-term solution by enabling mobile portals to adjust their structure according to the needs of individual users, in order to pre-empt their content needs and provide them with a more direct route to favoured content and services. We have described a particular approach to personalization to achieve this goal, but ultimately the success of this technique must be evaluated with respect to real users. On the face of it, the technique has the ability to significantly reduce portal click-distance, but concerns remain in relation to how live users might respond to a portal whose structure is changing, albeit gradually.

In this section we investigate the success of personalized navigation on live users. Specifically, we examine the hypothesis that user satisfaction will be significantly enhanced by reducing the click-distance of a mobile portal, on a user-by-user basis. In turn, by enhancing user satisfaction we expect to experience a significant increase in portal usage levels – users should spend more time online, returning to the portal more frequently to visit more content sites.

The following evaluation to test the above hypothesis is based on live-user field trials on European WAP portals. The standard trial consisted of a 2-week profiling period in which no personalization takes place, but the behaviour of the users is monitored in order to profile their navigation patterns. The remaining 4 weeks are divided into two 2-week personalization periods. During this time profiling continues, but in addition, personalization is switched on so that users experience a new portal structure that is adapted to their navigation preferences. The reported trial consists of 130 trialists from a variety of backgrounds and with a range of mobile usage habits and handsets.

### 5.1 Click-Distance Reduction

First up is the issue of click-distance, and the question to be answered refers to the extent that click-distance can be reduced by our personalized navigation technique. Figure 6 illustrates how portal click-distance changes during the trial in terms of the average user click-distance from the home page to each user's three most frequently accessed sites. The results show that a starting click-distance of 13.88 for the static

portal drops by over 50% to 6.84 during the first personalization period and by a further 2% for the final period.

These results show two things: first that significant click-distance reductions are possible; and second, these reductions are realized rapidly, in this case after only two weeks of profiling, which corresponds to about 3–5 sessions per user. Clearly, these observations are encouraging in the sense that significant click-distance reductions are achievable within a reasonable time frame. However, whether or not these reductions actually translate into fundamental changes in usage remain to be seen, and such issues are the focus of the following set of experiments.

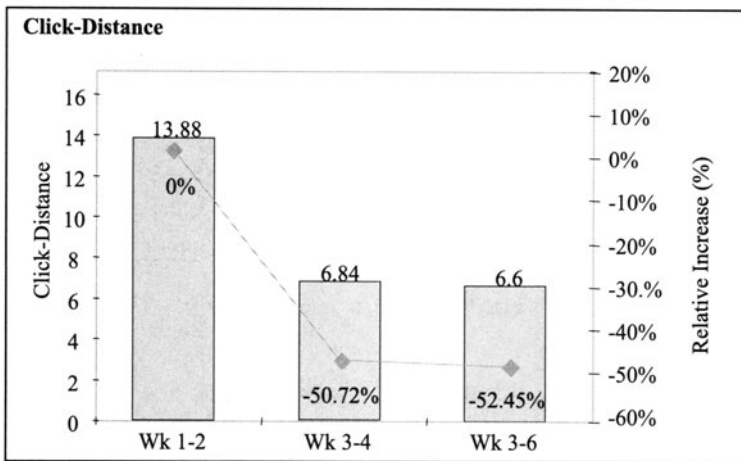


Fig. 6. Click-distance results

## 5.2 Navigation Time Versus Content Time

In the introduction to this article we distinguished between navigation time and content time in relation to mobile portal access, arguing that while content time brought value to the user, navigation time was essentially valueless since navigating to content is simply a means to an end. Given that the results above indicate a clear reduction in portal click-distance, we would expect that this translates into a comparable reduction in navigation time, but whether it has any impact on content time needs to be clarified.

First, Fig. 7 shows how the above click-distance reduction translates into a reduction in average daily navigation time. Over the 4-week personalization period (weeks 3–6) average daily navigation time was reduced by 36%. During the initial static period users spent an average of 56.42 seconds navigating to content each day, but this fell to only 35.99 seconds for the 4 weeks of personalization. Indeed, if we look at the results for the final two weeks (weeks 5–6) in comparison to the first two weeks of



personalization (weeks 3–4) we see the incremental benefits of personalization more clearly, with navigation time reducing from an average of 36.55 seconds (weeks 3–4) to 35.43 seconds (weeks 5–6).

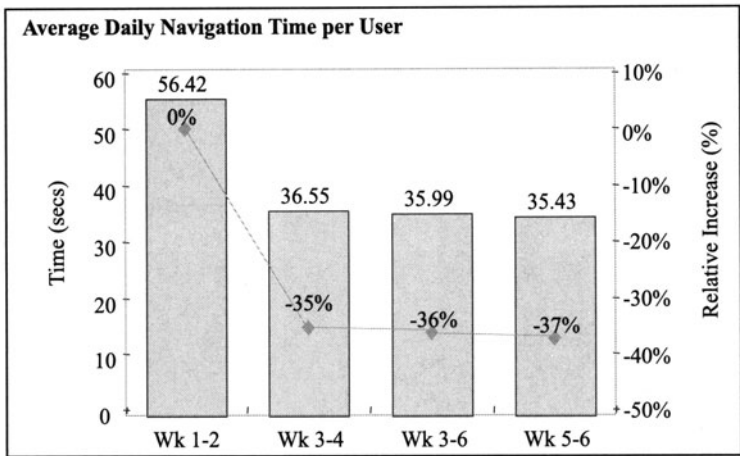


Fig. 7. Navigation time results

It is important to realize that the above results refer to total daily navigation time for the average user. However, since the number of sites that a user accesses may change day by day, the above navigation times do not provide an accurate picture of the average navigation time for an individual content site. Figure 8 presents this data by dividing the above navigation times by the average number of daily site hits for each period. They show a clearer picture of what is really happening to navigation time, which is seen to decrease by 50% as a result of personalization. During the static period the average user took nearly 32 seconds to navigate to an individual content site, but this fell to about 16 seconds as a result of personalization.

While the reduction in navigation time is to be expected, given the strong relationship that exists between click-distance and navigation time, the implications for content time are less predictable. For this reason it is particularly encouraging to find significant increases in content time due to personalization (Fig. 9). Over the 4-week personalization period (weeks 3–6) average daily content time increases by over 16%. During the static period the average total daily content time per trialist is 312.46 seconds compared to 364.55 seconds as an average of the 4-week personalization period. Moreover, if we look at the average content time for the final two trial weeks (as opposed to the final four weeks) we find a relative increase of more than 22% (average content time of 382.62 seconds). Thus, the relative increase in content time for the final two weeks of the trial (22.45%) has more than doubled in comparison to the first two weeks of personalization (10.89%); as personalization proceeds so too do the benefits increase.

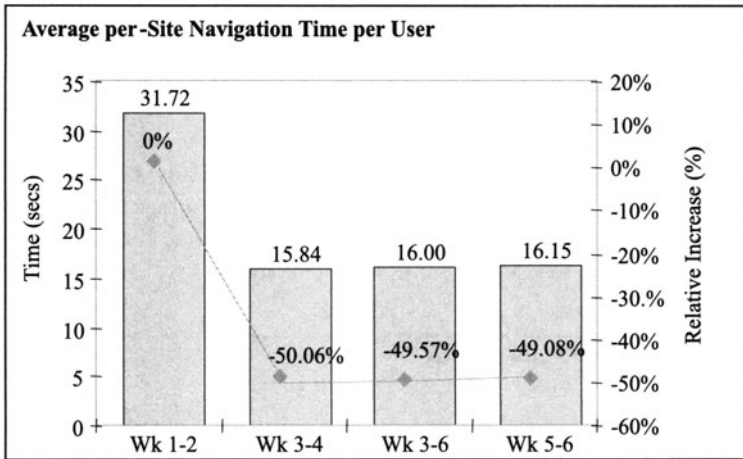


Fig. 8. Navigation time per site access

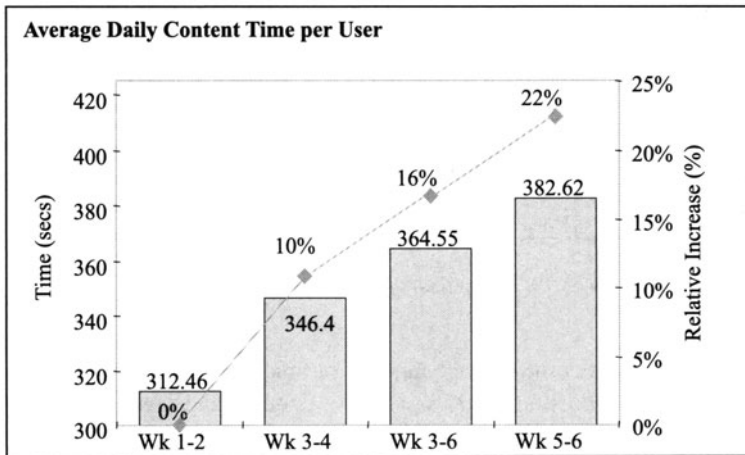


Fig. 9. Content time results

These results also highlight an important point about the willingness of users to trade savings in navigation time for increases in content time. According to these results, for every second of navigation time saved, the average user increases their content time by more than 3 seconds – by the final two weeks of personalization the average user is saving an average of 22.99 seconds in total navigation time and increases their total content time by 70.16 seconds. There are obvious benefits here for the mobile operator from a revenue point of view, not only in terms of existing airtime-based charging models but also as operators move to content-based charging models where navigation time charges must be eliminated or minimized, and so where it is critical to look for ways to reduce the need for navigation.

### 5.3 Site Hits

So far we have seen how users engage in additional content time as a result of personalization. How do they spend this extra time? For example, are they going to additional content sites or are they simply spending extra time in their usual favourites? The so-called *content discovery* problem refers to the challenge of how best to help users to discover new content services on a portal and how to encourage them to visit these services more frequently. It is especially important to mobile operators in order to maximise the return on investment for new content services.

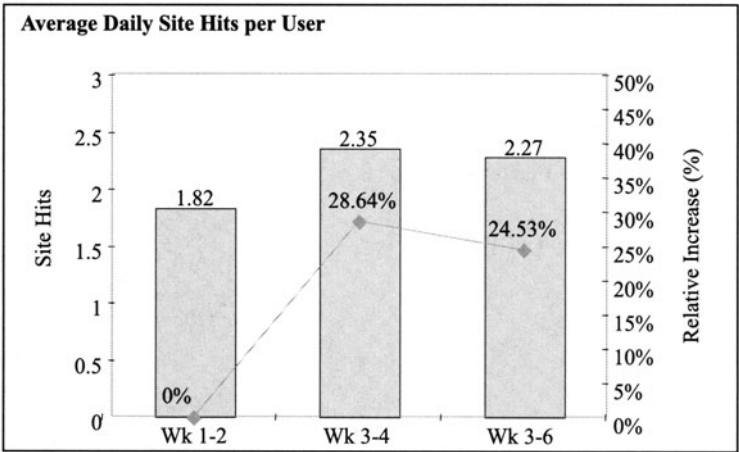


Fig. 10. The percentage of successful sessions

Figure 10 shows that the average number of unique site hits (individual visits to content sites) increased by nearly 25% when we compare the 4-week personalization period to the static period. In other words, users spent their extra time online going to extra content sites – the newfound ease with which users could access content led them to additional sites. And thus there is clear evidence to indicate that personalizing the navigation structure of a mobile portal promotes content discovery.

## 6 Conclusions

In general, limited usability and poor value-for-money are major contributing factors to the low levels of interest in the mobile Internet currently shown by the general public. These problems are closely aligned with the difficulty that users have in locating content on mobile portals. The bottom line is that they are easily frustrated by the amount of time that they are expected to devote to navigating to content through multiple layers of menus and icons.

For a variety of reasons this navigation problem is far more acute on mobile portals than on traditional Web portals. By their very nature, mobile devices are

limited by their small screens and restricted input capabilities, making it particularly cumbersome for users to manipulate the hierarchical menu structures that are used to organise content in a mobile portal. In addition, since mobile portals are organised according to a *one-size-fits-all* policy many compromises are made with respect to the placement of content services. This exacerbates the navigation problem for many individual users because popular items for one user may be located far from the portal home page if they are not popular in general.

## 6.1 Solving the Navigation Problem

In this research we have attempted to solve the above challenge by taking advantage of user profiling and personalization techniques in order to automatically adapt the structure of a mobile portal for an individual user, based on their usage history. We have presented the click-distance metric as a model of navigation effort in mobile portals and described a probabilistic personalization technique for restructuring a mobile portal by reordering and promoting menu options in a way that minimises click-distance for a given user. In turn, we have found that this strategy can reduce the average click-distance for a typical user by 50% in live-user trials.

Of course, to be successful it is not sufficient to simply reduce click-distance. The real issue is whether users perceive a usability benefit from this reduction. The evidence from extensive trials suggests that they do, and our studies indicate that significant benefits are available across a range of critical usage metrics (e.g., airtime and site hits). Indeed, for every second of navigation time that is saved users engage in an additional 3 seconds of content time.

Incidentally, the significance of reduced navigation time should not be underrated. By cutting navigation time in half, *perceived* bandwidth is effectively doubled without a major infrastructural investment. Moreover, this bandwidth doubling can be achieved on top of future bandwidth increases and, as operators upgrade to faster GPRS networks, personalized navigation can double the effective bandwidth here too.

It is worth pointing out that the personalization technique described here forms an important component of the ClixSmart Navigator platform developed by Changing-Worlds Ltd. ([www.changingworlds.com](http://www.changingworlds.com)). Moreover, while the results reported here are derived from a modest 6-week trial of 130 users, similar results have been repeatedly found for much larger trials and deployments (in excess of 500 thousand users over an 18-month period) with European operators. ClixSmart Navigator is today deployed by Europe's leading mobile operators, including Vodafone and O<sub>2</sub>.

## 6.2 Open Problems

Finally, in closing it is worth highlighting a number of open issues and challenges that remain in relation to mobile portal usability. Specifically, we focus on issues that are directly related to portal personalization.

**Multi-Criteria Personalization.** The techniques described in this work, for automatically adapting the structure of a mobile portal, are based on a probabilistic personalization strategy that relies primarily on the number of accesses a user makes to given menus and sites within the portal. However, in general, other factors are likely to be important; for example, recency of access and time of day may have a significant impact on accurately predicting the user's current interests. The issue of how best to incorporate these additional factors requires further attention. One approach is to look at ways of combining multiple factors into a single unified relevancy function, but of course the issue then becomes how such factors should be weighted and combined. For example, should recency of access be considered to be more important than frequency? Indeed, it is possible that different users may prefer different weighting schemes, which would add considerably to the complexity of any personalization strategy.

**Multiple Personalities.** It is also interesting to note that mobile users often display very different usage patterns as they use their mobile portal. For example, during working hours a user may focus on business-related services, and during leisure hours she may focus on entertainment information. More generally, this suggests that users can be associated with different usage *personalities* rather than a single all-encompassing profile. The ability to accurately identify and separately profile these personalities is likely to have a significant impact on the effectiveness of personalization. Once again, the issue of how best to model and capture these multiple personalities is a significant and challenging research issue. For example, automated clustering techniques could be used in order to partition the behaviour of an individual user into different personality profiles.

**Personalized Presentation.** Finally, the techniques described in this article have focused on a very direct form of personalization, from a presentation viewpoint; options are essentially presented as a list or grid and ordered according to their relevance. Alternative strategies might look at adapting the screen space or level of detail devoted to certain options. For example, very relevant options and services might be offered increased screen space as well as being positioned more prominently within the portal.

## References

1. H. Ahmadi, R.H. Katz, I.F. Akyildiz, and Z.J. Haas, editors. *Proc. Second Annual International Conference on Mobile Computing and Networking*. ACM Press, 1996.
2. I.F. Akyildiz, J.Y.B. Lin, R. Jain, V. Bharghavan, and A.T. Campbell, editors. *Proc. Eighth Annual International Conference on Mobile Computing and Networking*. ACM Press, 2002.
3. C. Anderson, P. Domingos, and D. Weld. Adaptive Web navigation for wireless devices. In *Proc. 17th International Joint Conference on Artificial Intelligence*, pages 879–884, 2001.
4. B. Awerbuch and D. Duchamp, editors. *Proc. First Annual International Conference on Mobile Computing and Networking*. ACM Press, 1995.

5. S. Berg, A.S. Taylor, and R. Harper. Mobile phones for the next generation: device designs for teenagers. In *Proc. Conference on Human Factors in Computing Systems*, pages 433–440. ACM Press, 2003.
6. D. Billsus, M.J. Pazzani, and J. Chen. A learning agent for wireless news access. In *Proceedings of Conference on Intelligent User Interfaces*, pages 33–36, 2000.
7. S. Bjork, L.E. Holmquist, J. Redstrom, I. Bretan, R. Danielsson, J. Karlgren, and K. Franzen. WEST: a Web browser for small terminals. In *Proc. 12th Annual ACM Symposium on User Interface Software and Technology*, pages 187–196. ACM Press, 1999.
8. G. Buchanan, S. Farrant, M. Jones, H. Thimbley, Marsden G., and M. Pazzani. Improving mobile internet usability. In *Proc. 10th World Wide Web Conference*, pages 673–680, 2001.
9. K. Cheverst, N. Davies, K. Mitchell, and A. Friday. Mobile-awareness: designing for mobile interactive systems. *ACM SIGGROUP Bulletin*, 22(1):8–11, 2001.
10. M. Devarakonda, A. Joshi, and M. Viveros, editors. *Proc. First International Workshop on Mobile Commerce*. ACM Press, 2001.
11. M. Dunlop and S. Brewster. The challenge of mobile devices for human computer interaction. *Personal and Ubiquitous Computing*, 6(4):235–236, 2002.
12. X. Fu, J. Budzik, and K. Hammond. Mining navigation history for recommendation. In *Proceedings of Conference on Intelligent User Interfaces*, pages 106–112, 2000.
13. M. Fulk. Improving Web browsing on handheld devices. In *CHI '01 Extended Abstracts on Human Factors in Computer Systems*, pages 395–396. ACM Press, 2001.
14. E. Herder. Modelling user navigation. In *Proc. 9th International Conference on User Modelling*, pages 417–419. Springer, Berlin Heidelberg New York, 2003.
15. J. Jacko and G. Salvendy. Hierarchical menu design: Dreadth, depth and task complexity. *Perceptual and Motor Skills*, 82:1187–1201, 1996.
16. S. Jenson, A. Wagner, and A. Hodges. Exploding wireless myths: exploring the UI issues underlying the marketing hype. In *CHI '01 Extended Abstracts on Human Factors in Computer Systems*, pages 211–212. ACM Press, 2001.
17. A. Kaikkonen and V. Roto. Navigating in a mobile XHTML application. In *Proc. Conference on Human Factors in Computing Systems*, pages 329–336. ACM Press, 2003.
18. T. Kamba, S.A. Elson, T. Harpold, T. Stamper, and P. Sukaviriya. Using small screen space more efficiently. In *Proc. SIGCHI Conference on Human Factors in Computing Systems*, pages 383–390. ACM Press, 1996.
19. J.I. Kiger. The depth/breadth tradeoff in the design of menu-driven interfaces. *International Journal of Man/Machine Studies*, 20:201–213, 1984.
20. H. Kodesh, V. Bahl, T. Imielinski, and M. Steenstrup, editors. *Proc. Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking*. ACM Press, 1999.
21. L. Kärkkäinen and J. Laarni. Designing for small display screens. In *Proc. Second Nordic Conference on Human-Computer Interaction*, pages 227–230. ACM Press, 2002.
22. K. Larson and M. Czerwinski. Web page design: Implications of memory, structure and scent for information retrieval. In *Proc. CHI'98 Human Factors in Computer Systems*, pages 25–32. ACM Press, 1998.
23. E. Lee and J. MacGregor. Minimizing user search time in menu retrieval systems. *Human Factors*, 27:157–162, 1985.
24. G. Marsden, R. Cherry, and A. Haefele. Small screen access to digital libraries. In *CHI '02 Extended Abstracts on Human Factors in Computer Systems*, pages 786–787. ACM Press, 2002.

25. G. Marsden and M. Jones. Ubiquitous computing and cellular handset interfaces: are menus the best way forward? In *Proceedings of The South African Institute of Computer Scientists and Information Technologists Annual Conference (SAICSIT)*, pages 111–119, 2001.
26. D.P. Miller. The depth/breadth tradeoff in hierarchical computer menus. In *Proc. 25th Annual Meeting of the Human Factors and Ergonomics Society*, pages 296–300, 1981.
27. W.P. Osborne and D. Moghe, editors. *Proc. Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*. ACM Press, 1998.
28. L. Palen and M. Salzman. Beyond the handset: designing for wireless communications usability. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 9(2):125–151, 2002.
29. L. Pap, K. Sohrawy, D.B. Johnson, and C. Rose, editors. *Proc. Third Annual ACM/IEEE International Conference on Mobile Computing and Networking*. ACM Press, 1997.
30. M. Perkowitz. *Adaptive Web Sites: Cluster Mining and Conceptual Clustering for Index Page Synthesis*. PhD Thesis, Department of Computer Science and Engineering. University of Washington, 2001.
31. M. Perkowitz and O. Etzioni. Towards adaptive Web sites: Conceptual framework and case study. *Journal of Artificial Intelligence*, 18(1-2):245–275, 2000.
32. M. Perry, K. O'hara, A. Sellen, B. Brown, and R. Harper. Dealing with mobility: understanding access anytime, anywhere. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 8(4):323–347, 2001.
33. R. Pickholtz, S.K. Das, R. Caceres, and J.J. Garcia-Luna-Aceves, editors. *Proc. Sixth Annual International Conference on Mobile Computing and Networking*. ACM Press, 2000.
34. M. Ramsey and J. Nielsen. *The WAP Usability Report*. Neilsen Norman Group, 2000. <http://www.nngroup.com/reports/wap>.
35. D. Reiken. Special issue on personalization. *Communications of the ACM*, 43(8), 2000.
36. C. Rose, editor. *Proc. Eleventh Annual International Conference on Mobile Computing and Networking*. ACM Press, 2001.
37. A.J. Sellen and R. Murphy. The future of the mobile internet: Lessons from looking at Web use. *Appliance Journal*, 2002.
38. A.J. Sellen, R. Murphy, and K.L. Shaw. How knowledge workers use the Web. In *Proc. SIGCHI Conference on Human Factors in Computing Systems*, pages 227–234. ACM Press, 2002.
39. B. Smyth. *The Plight of the Mobile Navigator*. Mobile Metrix, 2002.
40. B. Smyth and C. Cotter. Wapping the Web: A case-study in content personalization for WAP-enabled devices. In *Proc. 1st International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'00)*, pages 98–108, 2000.
41. B. Smyth and P. Cotter. Personalized adaptive navigation for mobile portals. In *Proc. 15th European Conference on Artificial Intelligence - Prestigious Applications of Artificial Intelligence*. IOS Press, 2002.
42. B. Smyth and P. Cotter. The plight of the navigator: Solving the navigation problem for wireless portals. In *Proc. 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'02)*, pages 328–337. Springer, Berlin Heidelberg New York, 2002.
43. K. Snowberry, S.R. Parkinson, and N. Sisson. Computer display menus. *Ergonomics*, 26:699–712, 1983.
44. P. Thomas and R.D. Macredie. Introduction to the new usability. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 9(2):69–73, 2002.

45. Luca Chittaro and P. Dal Cin. Evaluating interface design choices on WAP phones: Navigation and selection. *Personal and Ubiquitous Computing*, 6(4):237–244, 2002.
46. M. Viveros, H. Lei, and O. Wolfson, editors. *Proc. Second International Workshop on Mobile Commerce*. ACM Press, 2002.
47. D. Wallace, N. Anderson, and B. Shneiderman. Time stress effects on two menu selection systems. In *Proc. 31st Annual Meeting of the Human Factors and Ergonomics Society*, pages 727–731, 1987.
48. S. Waterson, J.A. Landay, and T. Matthews. In the lab and out in the wild: remote Web usability testing for mobile devices. In *CHI '02 Extended Abstracts on Human Factors in Computer Systems*, pages 796–797. ACM Press, 2002.
49. J.M. Webb and A.F. Kramer. Maps or analogies? a comparison of instructional aids for menu navigation. *Human Factors*, 32:251–266, 1990.
50. P. Zaphirs. Depth vs. breadth in the arrangement of Web links. In *Proceedings of 44th Annual Meeting of the Human Factors and Ergonomics Society*, pages 139–144, 2000.



---

# Learning Web Request Patterns

Brian D. Davison

Department of Computer Science and Engineering, Lehigh University  
19 Memorial Drive West, Bethlehem, PA 18015 USA  
davison@lehigh.edu

**Summary.** Most requests on the Web are made on behalf of human users, and like other human-computer interactions, the actions of the user can be characterized by identifiable regularities. Much of these patterns of activity, both within a user and between users, can be identified and exploited by intelligent mechanisms for learning Web request patterns. Our focus is on Markov-based probabilistic techniques, both for their predictive power and for their popularity in Web modeling and other domains. Although history-based mechanisms can provide strong performance in predicting future requests, performance can be improved by including predictions from additional sources.

In this chapter we review the common approaches to learning and predicting Web request patterns. We provide a consistent description of various algorithms (often independently proposed) and compare performance of those techniques on the same data sets. We also discuss concerns for accurate and realistic evaluation of these techniques.

## 1 Introduction

Modeling user activities on the Web has value both for content providers and consumers. Consumers may appreciate better responsiveness as a result of precalculating and of preloading content into a local cache in advance of their requests. A user requesting content that can be served by the cache is able to avoid the delays inherent in the Web, such as congested networks and slow servers. Additionally, consumers may find adaptive and personalized Web sites that can make suggestions and improve navigation to be useful. Likewise, the content provider will appreciate the insights that modeling can provide and the financial benefits of a happier consumer that gets the desired information even faster.

Most requests on the Web are made on behalf of human users, and like other human-computer interactions, the actions of the user can be characterized as having identifiable regularities. Much of these patterns of activity, both within a user and between users, can be identified and exploited by intelligent mechanisms for learning Web request patterns.

Prediction here is different from what data mining approaches do with Web logs. We wish to build a (relatively) concise model of the user so as to be able to dynamically

predict the next action(s) that the user will take. Data mining of Web logs, in contrast, is typically concerned with characterizing the user, finding common attributes of classes of users, and predicting future actions (such as purchases) without the concern for interactivity or immediate benefit (e.g., see the KDD Cup 2000 competition [8]).

Therefore we might consider the application of machine learning techniques [44] to the problem of Web request sequence prediction. In particular, we wish to be able to predict the next Web page that a user will select. This chapter will demonstrate the use of machine learning models on real-world traces with predictive accuracies of 12–50% or better, depending on the trace.

Sequence prediction in general is a well-studied problem, particularly within the data compression field [4, 19, 64]. Unfortunately, some in the Web community have rediscovered many of these techniques, leading to islands of similar work with dissimilar vocabulary. Here we will both reexamine these techniques as well as offer modifications motivated by the Web domain. This chapter will describe, implement, and experimentally evaluate a number of methods to model usage and predict Web requests. Our focus will be on Markov-based and Markov-like probabilistic techniques, both for their predictive power, but also for their popularity in Web modeling and other domains.

Prediction can be applied to various types of Web workloads – those seen by clients, proxies, and servers. Each location provides a different view of Web activities, and the context in which they occur. As a result, different levels of performance will be possible. We will also briefly consider information retrieval techniques to allow the use of the content of Web pages to help predict future requests. Although history-based mechanisms can provide strong performance in predicting future requests, we will find that performance can be improved by including predictions from additional sources. Our contributions in this chapter include the consistent description of various algorithms (often independently proposed), the development and utilization of generalized prediction codes to implement some of those techniques, and consistent comparison of the performance of those techniques across data sets.

In the next section, we will detail the many concerns for accurate and realistic performance assessment and describe the approaches we will take in the empirical evaluation of various Web request prediction algorithms. In Sect. 3 we review the common approaches to learning and predicting Web request patterns. In Sect. 4 we describe the workloads used by our system (which is summarized in Sect. 5). We present and discuss experimental results in Sects. 6 and 7. An alternative to history-based prediction is proposed in Sect. 8. Section 9 reviews the findings of this chapter.

## 2 Evaluation Concerns and Approaches

Because much of the existing work on learning Web request patterns has been performed by researchers in many disciplines, we first discuss the various aspects of how and what to evaluate when we compare Web request prediction algorithms. Two high-level concerns that we address are the questions of whether to modify typical

evaluation approaches to better fit the domain, and whether to modify predictions to better fit the domain, or both.

## 2.1 Type of Web Logs Used

One important aspect of any experimental work is the data sets used in the experiments. While we will introduce the Web workloads that we will use in Sect. 4, the type of workload is an evaluation issue. At a high level, we are simply concerned with methods that learn models of typical Web usage. However, at a lower level, those models are often simply identifying co-occurrences among resources – with the ultimate goal to make accurate predictions for resources that might be requested given a particular context.

However, there are multiple types of relationships between Web resources that might cause recognizable co-occurrences in Web logs [6, 18, 16]. One possible relationship is that of an embedded object and its referring page – an object, such as an image, audio, or Java applet that is automatically retrieved by the browser when the referring page is rendered. Another relationship is that of traversal, that is, when a user clicks on a link from the referring page to another page. The first (embedding) is solely an aspect of how the content was prepared. The second (traversal), while likewise existing because of a link placed by the content creator, is also a function of how users navigate through the Web hypertext.

Many researchers (see, for example, [36, 45, 49, 69, 61, 59, 66, 11]) distinguish between such relationships and choose not to make predictions for embedded resources. They are concerned with click-stream analysis – just the sequence of requests made by the user and not the set of automatically requested additional resources. Sometimes the data can provide the distinction for us since a Web server often records the referrer in its access logs, which captures the traversal relationship. But the HTTP referrer header is not a required field and is thus not always available. In other cases, analysis of the data is required to label the kind of request, for example, requests for embedded links are usually highly concentrated in time near the referring page. Unfortunately, many of the publicly available access logs do not provide sufficient detail to allow us to make such distinctions conclusively. In addition, methods that use click-stream data exclusively will require a more complex implementation, as they assume that the embedded resources will be prefetched automatically, which requires parsing and may miss resource retrievals that are not easily parsed (such as those that result from JavaScript or Java execution). As a result, in this chapter we have chosen to disregard the type of content and the question of whether it was an embedded resource or not, and instead use all logged requests as data for training and prediction. This approach is also taken by Bestavros et al. [6, 5], and Fan et al. [28].

## 2.2 Per-User or Per-Request Averaging

One can calculate average performance on a per-user (sometimes termed macroaverage) or per-request (microaverage) basis. Macroaverage performance treats all users

equally, even though some users will be more active and generate more traffic than others. In contrast, microaverage performance emphasizes the requests made by highly active users.

While predictions in the approaches we examine are made on a per-user basis (that is, on the basis of that user's previous request, which is not necessarily the most recent request in the system), we do not always build per-user predictive models. Individual models of behavior require more space and tend to be less accurate because they see less data than a global model. In our experiments, we will build global models (which can be thought of as modeling the typical user) for Web servers and proxies, and we will only consider per-user models when making predictions at the client. Thus for comparison, we will report only per-request averages.

## 2.3 User Request Sessions

A session is a period of sustained Web activity by a user. In most traces, users have request activities that could be broken into sessions. In practice, it may be helpful to mark session boundaries to learn when not to prefetch, but alternately it may be desirable to prefetch the first page that the user will request at the beginning of the next session. We have not analyzed the data sets for sessions; for the purposes of this chapter, each trace is treated like a set of per-user strings of tokens. Thus, even though a Web log contains interleaved requests by many clients, our algorithms will consider each prediction for a client solely in the context of the requests made by the same client. In addition, it does not matter how much time has passed since the previous request by the same client, nor what the actual request was. If the next request received matches the predicted request, it is considered a success.

## 2.4 Batch Versus Online Evaluation

The traditional approach to machine learning evaluation is the batch approach, in which data sets are separated into distinct training and test sets. The algorithm attempts to determine the appropriate model by learning from the training set. The model is then used statically on the test set to evaluate its performance. This approach is used in a number of Web prediction papers [69, 1, 45, 57, 61, 66, 67, 65, 11]. While we can certainly do the same (and will do so in one case for system validation), our normal approach will be to apply the predictive algorithms incrementally, in which each request serves to update the current user model and assist in making a prediction for the next request. This matches the approach taken in other Web prediction papers [47, 48, 28, 49]. This model is arguably more realistic in that it matches the expected implementation, that is, a system that learns from the past to improve its predictions in the future. Similarly, under this approach we can test the code against all requests (not just a fraction assigned to be a test set), with the caveat that performance is likely to be poor initially before the user model has acquired much knowledge (although some use an initial warming phase in which evaluation is not performed to alleviate this effect).

When using the prediction system in a simulated or actual system, note that the predictions may cause the user's actual or perceived behavior to change. In prefetching, the user may request the next document faster if there was no delay in fetching the first (because it was preloaded into a cache). Likewise, a proxy- or server-based model that sends hints about what to prefetch or content to the browser will change the reference stream that comes out of the browser, since the contents of the cache will have changed. Thus, the predictive system may have to adjust itself to the change in activity that was caused by its operation. Alternatively, with appropriate HTTP extensions, the browser could tell the server about requests served by the cache (as suggested in [28, 26]). While it might be helpful to incorporate caching effects, in this chapter we limit ourselves to models that are built from the same requests as those that are used for evaluation. This model is appropriate for prefetching clients and proxies, as long as they do not depend on server hints (built with additional data).

## 2.5 Selecting Evaluation Data

Even when using an online per-user predictive model, it will be impossible for a proxy to know when to “predict” the first request, since the client had not connected previously. Note that if the predictions are used for server-side optimization, then a generic prefetch of the most likely first request for any user may be helpful and feasible, and similarly for clients a generic preloading of the likely first request can be helpful. Likewise, we cannot test predictions made after the user's last request in our sample trace. Thus, the question of which data to use for evaluation arises. While we will track performance along many of these metrics, we will generally plot performance on the broadest metric: the number of correct predictions out of the total number of requests. This is potentially an underestimate of performance, as the first requests and the unique requests (that is, those requests that are never repeated) are counted, and may become less of a factor over time. If we were to break the data into per-user sessions, this would be a larger factor as there would be more first and last requests that must be handled.

## 2.6 Confidence and Support

In most real-world implementation scenarios, there is some cost for each prediction made. For example, the cost can be cognitive if the predictions generate increased cognitive load in a user interface. Or the cost can be financial, as in prefetching when there is a cost per byte retrieved. Thus, we may wish to consider exactly when we wish to make a prediction, in other words, to refrain from taking a guess at every opportunity.

We consider two mechanisms to reduce or limit the likelihood of making a false prediction. They are:

- *Thresholds on confidence.* Confidence is loosely defined as the probability that the predicted request will be made, and is typically based on the fraction of the number of times that the predicted request occurred in this context in the past. Our

methods use probabilistic predictors, and thus each possible prediction has what can be considered an associated probability. By enforcing a minimum threshold, we can restrict the predictions to those that have high expected probability of being correct. Thresholds of this type have been used previously [47, 48, 41, 25, 65, 11].

- *Thresholds on support.* Support is strictly the number of times that the predicted request has occurred in this context. Even when a prediction probability (i.e., confidence) is high, that value could be based on only a small number of examples. By providing a minimum support, we can limit predictions to those that have had sufficient experience to warrant a good prediction [36, 58, 41, 28, 55, 61, 25, 66, 65].<sup>1</sup>

Typically, these two factors are combined in some function.

## 2.7 Calculating Precision

Given the ability to place minimum thresholds on confidence and support, the system may choose to refrain from making a prediction at all. Thus, in addition to overall accuracy (correct predictions/all requests), we will also calculate precision: the accuracy of the predictions when predictions are made. However, the exact selection of the denominator can be uncertain. We note at least two choices: those requests for which a prediction was attempted, and those requests against which a prediction was compared. The former includes predictions for requests that were never received (i.e., made beyond the last request received from a client). Since we cannot judge such predictions, when reporting precision, we will use the latter definition.

## 2.8 Top- $n$ Predictions

All of the variations we have described above provide probabilities to determine a prediction. Typically there are multiple predictions possible, with varying confidence and support. Since it may be useful (and feasible) to make predictions for more than one object simultaneously, we will explore the costs and benefits of various sets of predictions. As long as additional predictions still exceed minimum confidence and support values, we will generate them, up to some maximum prediction list length of  $n$ . The top- $n$  predictions can all be used for prediction (and, for example, prefetching), and depending on the evaluation metric, it may not matter which one is successful, as long as one of the  $n$  predictions is chosen by the user.

In some systems (e.g., [28]), there are no direct limits to the number of predictions made. Instead, effective limits are achieved by thresholds on confidence or support, or by available transmission time when embedded in a prefetching system. In this chapter we do not directly limit the number of predictions, but instead consider various threshold values. Limits are typically needed to trade off resources expended versus benefits gained, and that trade-off depends on the environment within which the predictions are being made.

<sup>1</sup> In fact, Su et al. [61] go further, requiring thresholds not only of page popularity (i.e., support) but also ignoring sequences below some minimum length.

B		
3	1	2

Data recorded is 'B'. The first number (3) is the number of times 'B' has been seen in this context (SelfCount). The second (1) is the count of the number of children of this node (NumChildren). The third (2) is the total number of times its children have been seen (ChildCounts).

**Fig. 1.** A sample node in a Markov tree

### 3 Prediction Techniques

In this section we describe a number of prediction algorithms and their variations used by Web researchers.

#### 3.1 *n*-Grams and Markov Models

Typically the term *sequence* is used to describe an ordered set of actions (Web requests, in this case). Another name, from statistical natural language processing, for the same ordered set is an *n*-gram. Thus, an *n*-gram is a sequence of *n* items. For example, the ordered pair  $(A, B)$  is an example 2-gram (or bigram) in which *A* appears first, followed by *B*.

For prediction, we would try to match the complete prefix of length  $n - 1$  (i.e., the current context) to an *n*-gram, and predict the *n*th request based on the last item of the *n*-gram. Since there may be multiple *n*-grams with the same prefix of  $n - 1$  requests, and *n*-grams do not natively provide the means to track their frequency, a mechanism is needed to determine which *n*-gram (of those matching the  $n - 1$  request prefix) should be used for prediction.

Markov models provide that means, by tracking the likelihood of each *n*-gram in a state space encoding the past. In this approach, we explicitly make the Markov assumption, which says that the next request is a function strictly of the current state. In a *k*-step Markov model, then, each state represents the sequence of *k* previous requests (the context), and has probabilities on the transitions to each of the next possible states. Since *k* is fixed, there are at most  $|a|^k$  states in such a system (where  $|a|$  is the number of possible requests).

Each state in the Markov model corresponds to the sequence of  $k = n - 1$  requests that comprise the prefix of an *n*-gram. The *n*th element of the *n*-gram (the next request) determines the destination of a link from this state to a future state, with a label corresponding to the last  $n - 1$  requests in the *n*-gram. Thus a Markov model can encompass the information in multiple *n*-grams, but additionally tracks the likelihood of moving from one state to another.

Typically *k* is fixed and is often small in practice to limit the space cost of representing the states. In our system we allow *k* to be of arbitrary size, but in practice we will typically use relatively small values (primarily because long sequences are infrequently repeated on the Web).

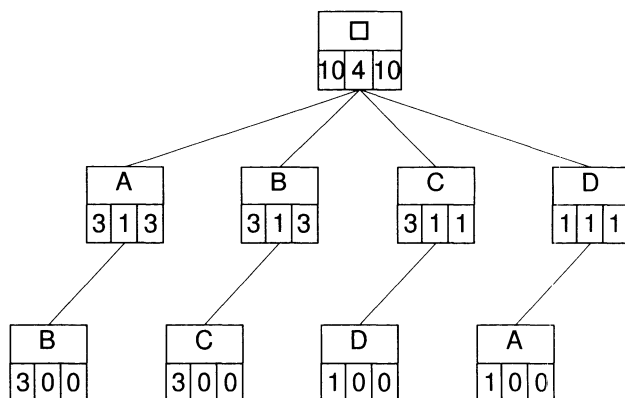
#### 3.2 Markov Trees

A more capable mechanism for representing past activity in a form usable for prediction is a Markov tree [60, 40, 27] in which the transitions from the root node to its

children represent the probabilities in a zeroth-order Markov model, the transitions to their children correspond to a first-order model, and so on. The tree itself thus stores sequences in the form of a trie, that is, a data structure that stores elements in a tree, where the path from the root to the leaf is described by the key (the sequence, in this case). A description of an individual node is shown in Fig. 1.

Figure 2 depicts an example Markov tree of depth two with the information recorded after having two visitors with the given request sequences. The root node corresponds to a sequence without context (that is, when nothing has come before). Thus, a zeroth-order Markov model interpretation would find the naive probability of an item in the sequence to be .3, .3, .3, and .1, for items A, B, C, and D, respectively. Given a context of B, the probability of a C following in this model is 1. These probabilities are calculated from the node's *SelfCount* divided by the parent's *ChildCounts*.

To make this process clear, pseudocode to build a Markov tree is provided in Fig. 3, and Fig. 4 illustrates one step in that process. Given the sequence (A, B, A), the steps taken to update the tree are described in Fig. 4a to get the tree in Fig. 4b. All of the suffixes of this sequence will be used, starting with the empty sequence. Given a sequence of length zero, we go to the root and increment *SelfCount*. Given next the sequence of length one, we start at the root and update its *ChildCounts*, and since there is already a child A, update that node's *SelfCount*. Given next the sequence of length two, we start again from the root, travel to child B, and update its *SelfCount* and find that we need to add a new child. Thus we also update B's *NumChildren*, and its *ChildCounts* as we add the new node. Assuming we are limiting this tree to depth two, we have finished the process, and have the structure shown in Fig. 4b.



Sequences from two different sessions: (A, B, C, D, A, B, C) followed by (A, B, C)

**Fig. 2.** A sample trace and simple Markov tree of depth two built from it



Given a sequence  $s$ , a  $\text{MaxTreeDepth}$ , and an initial tree (possibly just a root node)  $t$ :

```

for  $i$  from 0 to  $\min(|s|, \text{MaxTreeDepth})$ 
  let  $ss$  be the subsequence containing the last  $i$  items from  $s$ 
  let  $p$  be a pointer to  $t$ 
  if  $|ss| = 0$ 
    increment  $p.\text{SelfCount}$ 
  else
    for  $j$  from  $\text{first}(ss)$  to  $\text{last}(ss)$ 
      increment  $p.\text{ChildCounts}$ 
      if not-exists-child( $p, j$ )
        increment  $p.\text{NumChildren}$ 
        add a new node for  $j$  to the list of  $p$ 's children
      end if
      let  $p$  point to child  $j$ 
      if  $j = \text{last}(ss)$ 
        increment  $p.\text{SelfCount}$ 
      end if
    end for
  end if
end for

```

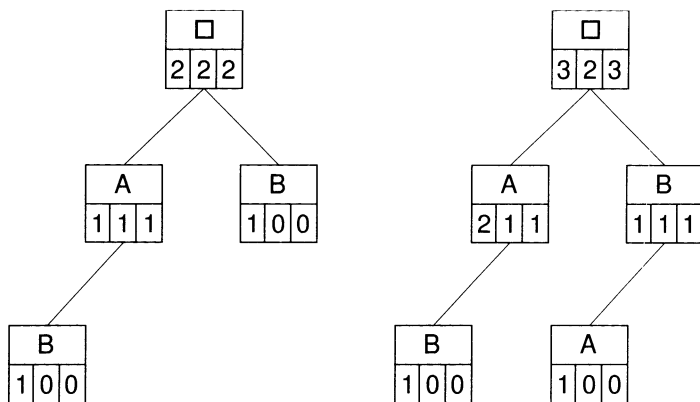
**Fig. 3.** Pseudocode to build a Markov tree

### 3.3 Path and Point Profiles

Thus, after building a Markov tree of sufficient depth, it can be used to match sequences for prediction. Schechter et al. [58] describe a predictive approach that is effectively a Markov tree similar to what we have described above. The authors call this approach using *path profiles*, a name borrowed from techniques used in compiler optimization, as contrasted with *point profiles*, which are simply bigrams (first-order Markov models). The longest path profile matching the current context is used for prediction, with frequency of occurrence used to select from equal-length profiles.

### 3.4 $k$ th Order Markov Models

Our Markov tree can, in general, be used to find  $k$ th order Markov probabilities by traversing the tree from the root in the order of the  $k$  items in the current context. Many researchers (e.g., [47, 48, 6, 5, 36, 45, 41, 26, 14, 62, 29, 67]) have used models roughly equivalent to first-order Markov models (corresponding to trees like that depicted in Fig. 2) for Web request prediction, but they are also used in many other domains (e.g., UNIX command prediction [23] and hardware-based memory address prefetching [37]). Others have found that second-order Markov models give better predictive accuracy [59, 69], and some, even higher order models (e.g., fourth-order [53]).



(a) Markov tree from sequence (A, B) (b) Markov tree from sequence (A, B, A)

**Fig. 4.** Before and after incrementally updating a simple Markov tree

During the use of a nontrivial Markov model of a particular order, it is likely that there will be instances in which the current context is not found in the model. Examples of this include a context shorter than the order of the model, or contexts that have introduced a new item into the known alphabet (that is, the set of requests seen so far). Earlier we mentioned the use of the longest matching sequence (path profile) for prediction. The same approach can be taken with Markov models. Given enough data, Markov models of high order typically provide high accuracy, and so using the largest one with a matching context is commonly the approach taken. Both Pitkow and Pirolli [55] and Deshpande and Karypis [25] take this route, but also consider variations that prune the tree to reduce space and time needed for prediction (e.g., to implement thresholds on confidence and support, testing on a validation set, and minimum differences in confidence between first and second most likely predictions). Su et al. [61] also combine multiple higher-order  $n$ -gram models in a similar manner. Interestingly, Li et al. [42] argue in contrast that the longest match is not always the best and provide a pessimistic selection method (based on Quinlan's pessimistic error estimate [56]) to choose the context with the highest pessimistic confidence of all applicable contexts, regardless of context length. They show that this approach improves precision as the context gets larger.

### 3.5 PPM

There are, of course, other ways to incorporate context into the prediction mode. PPM, or prediction by partial matching [4, 64], is typically used as a powerful method for data compression. It works similarly to the simple Markov model-based approach above, using the largest permissible context to encode the probabilities of the next item. However, it also tracks the probability of the next item to be something that has never been seen before in this context (called the “escape” symbol when used

Trace name	Described	Requests	Clients	Servers	Duration
EPA-HTTP	[46]	47748	2333	1	1 day
Music Machines	[51, 52]	530873	46816	1	2 months
SSDC	[24]	187774	9532	1	8 months
UCB-12days	[30, 31]	$5.95 \times 10^6$	7726	41836	12 days
UCB-20-clients	Sect. 4.2	163677	20	3755	12 days
UCB-20-servers	Sect. 4.3	746711	6620	20	12 days

**Table 1.** Traces used for prediction and their characteristics

for compression), and thus explicitly says to use the next smaller matching context instead. There are multiple versions of PPM that correspond to different ways of calculating the escape probability: PPM-A, PPM-C, PPM-D, as well as others. Since the next item in the sequence could (and is) given some probability at each of the levels below the longest matching context, all matching contexts must be examined to sum the probabilities for each candidate prediction (appropriately weighted by the preceding level's escape probability). This is accomplished by merging the current set of predictions with those from the shorter context by multiplying those probabilities from the shorter context by the escape symbol confidence ( $e$ ) in the longer context, and multiplying those in the longer context by  $(1 - e)$ . Various escape probabilities can then be calculated using the counts stored in our Markov tree.

A few researchers have used PPM-based models for Web prediction [49, 28, 10, 11]. Actually, Fan et al. [28] go further, building Markov trees with larger contexts. Instead of using large contexts (e.g., order  $n$ ) for prediction, they use a context smaller than  $n$  (say,  $m$ ), and use the remaining portion of the tree to make predictions of requests up to  $n - m$  steps in advance.

### 3.6 Additional Parameters

In addition to the varieties described above, the prediction system can also vary a number of other parameters that are motivated by the Web domain. One that we will explicitly test here is the size of the prediction window (that is, the window of requests against which the prediction is tested). Typically evaluation is performed by measuring the accuracy of predicting the next request. Instead of only predicting the very next request, we can measure the accuracy when the prediction can match any of the next  $n$  requests. This may be useful in matching the utility of preloading a cache as the resource may be useful later.

## 4 Prediction Workloads

There are three primary types of Web workloads, corresponding to three viewpoints on the traffic. Here we characterize each of the three and describe the data sets of each type that we will use. A summary of the datasets used can be found in Table 1.

## 4.1 Proxy

Typically sitting between a set of users and all Web servers, a proxy sees a more limited user base than origin Web servers, but in some cases a proxy may inherently group together users with some overlapping interests (e.g., users of a workgroup or corporate proxy may be more likely to view the same content). It typically records all requests not served by browser caches, but logs may contain overlapping user requests from other proxies or from different users that are assigned the same IP address at different times.

The models built by proxies can vary from the typical user in a single model to highly personalized models for each individual user. In any case, the predictive model can be used to prefetch directly into its cache, or to provide hints to a client cache for prefetching.

We will use one proxy trace in our experiments. The UC Berkeley (UCB) Home IP HTTP Traces [30] are a record of Web traffic collected by Steve Gribble as a graduate student in November 1996. Gribble used a snooping proxy to record traffic generated by the UC Berkeley Home IP dialup and wireless users (2.4 Kbps, 14.4 Kbps, and 28.8 Kbps land-line modems, and 20-30 Kbps bandwidth for the wireless modems). This is a large trace, from which we have selected the first 12 out of 18 days (for comparison with Fan et al. [28]), for a total of close to six million requests.

## 4.2 Client

The client is one of the two necessary participants in a Web transaction. A few researchers have recorded transactions from within the client browser [9, 17, 15, 63], making it possible to see exactly what the user does – clicking on links, typing URLs, using navigational aids such as back and forward buttons and bookmarks. It is also typically necessary to log at this level to capture activity that is served by the browser cache.

Using an individual client history to build a model of the client provides the opportunity to make predictions that are highly personalized, and thus reflect the behavior patterns of the individual user. Unfortunately, logs from augmented browsers are rare. Instead, a subset of requests captured by an upstream proxy (from the UCB data set) will be used with the understanding that such traces do not reflect all user requests, just those that were not served from the browser cache.

We have extracted individual request histories from the UCB proxy trace. However, not all ‘clients’ identified from proxy traces with unique IP addresses are really individual users. Since proxies can be configured into a hierarchy of proxy caches, we have to be concerned with the possibility that proxy traces could have “clients” which are really proxy caches themselves, with multiple users (or even proxies!) behind them. Likewise, even when a client corresponds to a particular nonproxy system, it may correspond to a mechanized process that repeatedly fetches one resource (or a small set of resources). Since the latter correspond to highly regular request patterns, and the former correspond to overlapping request patterns, we will attempt to avoid the worst of both in the concern for fairness in evaluation. We have thus ranked the

clients by total numbers of requests and ignored the top 20, and instead selected the second 20 as the representative set of active users.

### 4.3 Server

Servers provide a complementary view on Web usage from that of clients. Instead of seeing all requests made by a user, they see all requests made to one or more Web sites. Since they only know of requests for particular sites, such logs are unlikely to contain information about client transitions to other systems. Technically, the HTTP referrer header provides information on transitions into a particular site, but these are rarely provided in publicly available logs.

When learning a predictive model, the server could build an individual model for each user. This would be useful to personalize the content on the Web site for the individual user, or to provide hints to the client browser on what resources would be useful to prefetch. One difficulty is the relative scarcity of information unless the user visits repeatedly, providing more data than the typical site visit, which commonly contains requests for only a handful of resources. An alternative is to build a single model of the typical user – providing directions that may say that most users request resource B after fetching resource A. When trends are common, this approach finds them. A single model can also provide information that could be used in site re-design for better navigation [50, 52].

In between the two extremes lies the potential for a collaborative filtering approach in which individual models from many users can contribute to suggest requests useful to the current user, as pointed out by Zukerman et al. [68], or clustering of users (as in [65]). For the experiments in this chapter, we generally build a single model based on the traffic seen where the model is stored.<sup>2</sup> Thus, a server would build a single model, which while likely not corresponding to any particular user, would effectively model the typical behavior seen.

In addition to those already described, predictive Web server usage models can also be used to improve server performance through in-memory caching, reduced disk load, and reduced loads on back-end database servers. Similarly, they could be used for prefetching into a separate server-specific reverse proxy cache (with similar benefits). Server logs are widely available (relative to other kinds of logs), but they have some limitations, as they do not record requests to nonserver resources and do not see responses served by downstream caches (whether browser or proxy).

The experiments in this chapter use traces from three servers. The first is the EPA-HTTP server logs [46], which contain close to 48,000 requests corresponding to 24 hours of service at the end of August 1995. The second is two months of usage from the Music Machines Web site [51, 52], collected in September and October of 1997. Unlike most Web traces, the Music Machines Web site was specifically configured to prevent caching, so the log represents all requests (not just the browser cache misses). The third (SSDC) is a trace of approximately eight months of nonlocal usage of a

<sup>2</sup> However, the use of nontrivial contexts (such as a Markov model with order greater than 1) effectively clusters the user with other users who have experienced the same context.

Web site for a small software development company. This site was hosted behind a dedicated modem, and was collected over 1997 and 1998. Additionally, we have extracted the 20 most-popular servers from the UCB proxy trace.

## 5 Experimental System

In this section we describe the experimental prediction system that we use in subsequent experiments.

### 5.1 Implementation

To implement various sequence prediction methods, a highly parameterized prediction system was implemented in approximately 5000 lines of C code. We use the Markov tree data structure described in Sect. 3.2 since it works for the more complex algorithms, and just ignore some aspects of it for the simpler ones.

In effect, we have built essentially what Laird and Saul [39, 40] call a Transition Directed Acyclic Graph (TDAG). Like them, we limit the depth of the tree explicitly and limit the number of predictions made at once. In contrast, they additionally employ a mechanism to limit the expansion of a tree by eliminating nodes in the graph that are rarely visited.

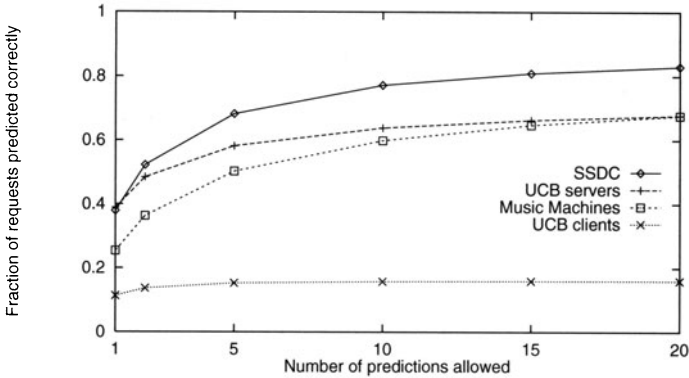
As mentioned in Sect. 3.4, confidence and support thresholds are believed to be useful, and so our code incorporates such thresholds and in general provides for a minimum and maximum  $n$ -gram length for prediction. Finally, for the purposes of these tests we will focus on potential predictive performance (e.g., accuracy) and ignore certain aspects of implementation efficiency. In particular, our codes are designed for generality, and not necessarily for efficiency.

### 5.2 Validation

In order to help validate our prediction codes, we replicated (to the extent possible) Sarukkai's HTTP server request prediction experiment [57]. This experiment used the EPA-HTTP data set, in which the first 40,000 requests were used as training data, and the remainder for testing.

We set up our tests identically and configured our prediction codes to use a first-order Markov model (i.e., an  $n$ -gram size of 2, with no minimum support or confidence needed to predict). Thus, unlike the remainder of the experiments presented in this chapter, this experiment builds a model using the initial training data and freezes it for use on the test data. This static prediction model corresponds closely to the performance of the first test of Markov chains reported by Sarukkai. We found an approximately 1% absolute increase in predictive accuracy of the same system when it is allowed to incrementally update its model as it moves through the test set.

The EPA-HTTP logs, however, are rather short (especially the test set), and so we consider the performance of prediction for other server logs in subsequent tests and figures. Since they provide a more realistic measure of performance, incremental predictive accuracy will be used throughout the rest of this chapter.



**Fig. 5.** Predictive accuracy for bigrams (first-order Markov models) under four data sets with varying numbers of allowed predictions

## 6 Experimental Results

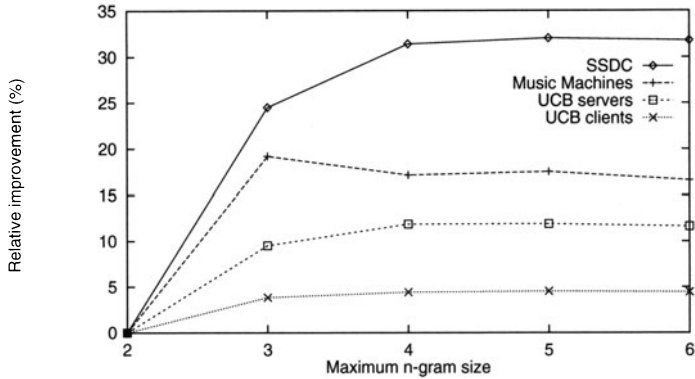
In this section we examine the effect of changes to various model parameters on predictive performance. In this way we can determine the sensitivity of the model (or data sets) to small and large changes in parameter values, and can find useful settings of those parameters for the tested data sets.

### 6.1 Increasing Number of Predictions

Depending on the situation in which the predictive system is embedded, it may be helpful to predict a set (Top- $n$ ) of possible next actions, as described in Sect. 2.8. In Fig. 5, we examine incremental predictive performance of simple bigrams while varying a single parameter (the number of predictions permitted) over four traces (SSDC, UCB servers, Music Machines, and UCB clients). For the initial case of one allowed prediction, we find that performance ranges from slightly over 10% to close to 40% accuracy. As the number of predictions allowed increases to 20, predictive performance increases significantly, which is a relative improvement of between 45% and 167%. The server-based traces show marked performance increases, demonstrating that the traces do indeed contain sufficient data to include most of the choices that a user might make. The performance of the client-based trace, conversely, remains low, demonstrating that the experience of an individual user is insufficient to include much of the activities that the user will perform in the future. Finally, we note that to achieve such high levels of predictive performance, systems will need to make many predictions, each of which come with some resource cost, such as time, bandwidth, and CPU usage.

### 6.2 Increasing $n$ -Gram Size

One potential way to improve accuracy is to consider  $n$ -grams larger than two. This increase in context allows the model to learn more specific patterns. Figure 6 shows



**Fig. 6.** Relative improvement in predictive accuracy for multiple data sets as maximum context length grows (as compared to a context length of 2)

the relative improvement (compared to  $n=2$ ) in incremental predictive accuracy for multiple traces when making just one prediction at each step. Thus, if the accuracy at  $n=2$  were 0.2, an accuracy of 0.3 would be a 50% relative improvement, and all traces are shown to have zero improvement at  $n=2$ . In each case, the longest  $n$ -gram is used for prediction (ties are broken by selecting the higher probability  $n$ -gram), and 1-grams are permitted (i.e., corresponding to predictions of overall request popularity) if nothing else matches. The graph shows that adding longer sequences does help predictive accuracy, but improvement peaks and then wanes as longer but rarer (and less accurate) sequences are used for prediction.<sup>3</sup> Thus, the figure shows that larger  $n$ -grams themselves can be useful, but if the largest  $n$ -grams were used alone, prediction performance would be significantly harmed. This is because longer  $n$ -grams match fewer cases than shorter  $n$ -grams, and thus are able to make fewer correct predictions.

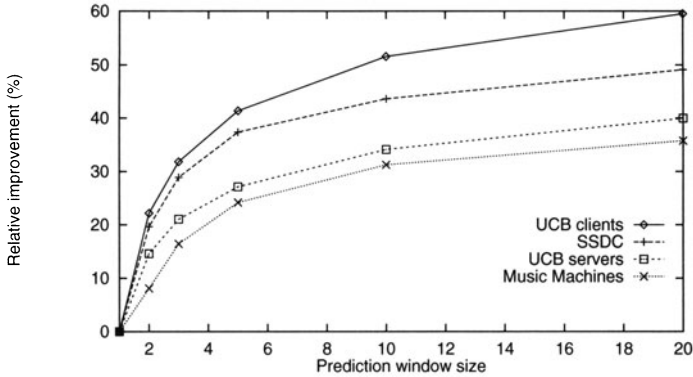
6.3 Incorporating Shorter Contexts

Prediction by partial match provides an automatic way to use contexts shorter than the longest matching one. In our experiments, we find that the PPM variations are able to perform slightly better than the comparable longest  $n$ -gram match. PPM-C gives the best results, with relative improvements ranging from 1–3% when only the best prediction is used. When multiple predictions are permitted, the performance improvement is even smaller.

However, we can also endow  $n$ -grams with the ability to incorporate shorter  $n$ -grams. In this alternative, the  $n$ -grams always merge with the results of prediction at the shorter context, with a fixed weight (as opposed to the weight from the dynamically calculated escape probability in the PPM model). Experiments reveal similar relative improvements of only a few percent.

<sup>3</sup> As a result, in subsequent tests we will often use an  $n$ -gram limit of 4, as the inclusion of larger  $n$ -grams typically does not improve performance.





**Fig. 7.** Relative improvement in predictive accuracy as the window of requests against which each prediction is tested grows

#### 6.4 Increasing Prediction Window

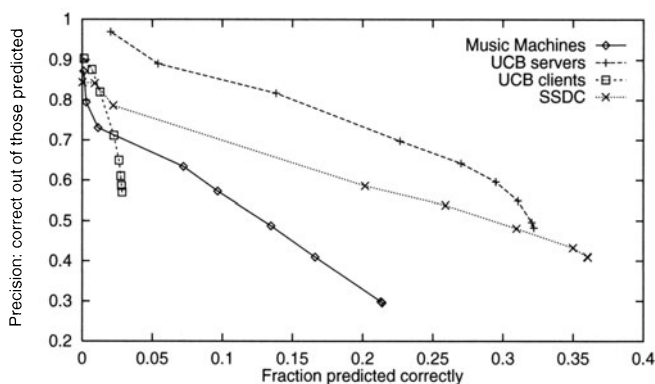
Another way to improve reported accuracy is to allow predictions to match more than just the very next request. In a real Web system, prefetched content would be cached and thus available to satisfy user requests in the future. One can argue that when a prediction does not match the next request, it is incorrect. However, if the prediction instead matches and can be applied to some future request, we should count it as correct. Thus, in this test we apply a Web-specific heuristic for measuring performance.

In Fig. 7 we graph the relative performance improvement that results when we allow predictions to match the next 1, 2, 3, 5, 10, and 20 subsequent requests. While at 20 requests performance continues to increase, the rate of growth is tapering. The apparent boost in potential performance suggests that even when the next request is not predicted perfectly, the predicted requests are potentially executed in the near future.

Predictive accuracy, even when using the mechanisms described here, is limited at least by the recurrence rates of the sequences being examined. That is, in this section we only consider predicting from history, which means that every unique request cannot be predicted when it is first introduced. Thus, if we were to plot predictive performance on a scale of what is possible, the graphs would be 2–6 absolute percentage points higher.

#### 6.5 Considering Mistake Costs

Accuracy alone is typically only important to researchers. In the real world, there are costs for mistakes. As the number of allowed predictions per request increases the fraction predicted correctly grows, but the number of predictions needed to get that accuracy also increases significantly. In a prefetching scenario in which mistakes are not kept for Music Machines or SSDC, the average per-request bandwidth could be



**Fig. 8.** Ratio of precision to overall accuracy for a minimum support of 10 with varying confidence.

up to twenty times the nonprefetching bandwidth. (Note, however, that this analysis does not consider the positive and extensive effects of caching.) The important point is that there is always a tradeoff – in this case, for increasing coverage we will consume additional bandwidth.

Therefore, it is worthwhile to consider how to reduce or limit the likelihood of making a false prediction, as discussed in Sect. 2.6. Increased precision is possible (with lower overall accuracy) as the threshold is increased. Likewise, while larger values of precision are possible, they come at a cost of the number of requests predicted correctly.

Figure 8 provides a combined view: a minimum support of 10 instances combined with a varying confidence threshold. Ideal performance is up and to the right (high precision and accuracy). In this figure, this combination (of minimum support of 10 along with some varying threshold on confidence) is able to achieve precision that exceeds that possible from either threshold alone. The lesson here is that high accuracy predictions are quite achievable, but are typically applicable to a much smaller fraction of the trace.

## 7 Discussion

We have demonstrated various techniques to improve predictive accuracy on logs of Web requests. Most of these methods can be combined, leading to significant improvements. A first-order Markov model is able to get approximately 38% top-1 predictive accuracy on the SSDC trace. A top-5 version can increase that by another 30% to have 68% predictive accuracy. After a few more adjustments (including using PPM, version C, with maximum  $n$ -gram 6, reducing predictions likely to be in cache down to 20%, and including past predictions with weight 0.025), accuracies of 75% are possible. Put another way, this constitutes a 20% reduction in error. And if we consider only those predictions that were actually tested (i.e., ignoring those for which the client made no more requests), we get a prediction accuracy of over 79%.

This performance, of course, comes at the cost of a large number of guesses. In this case, we are making five guesses almost all of the time. Alternatively, lower overall accuracy but at higher precision is possible with some modifications (such as the use of confidence and support thresholds).

Some investigators have focused on this issue. Although they use a simplistic model of co-occurrence, Jiang and Kleinrock [36] develop a complex cost-based metric (delay cost, server resource cost) to determine a threshold for prefetching, and use it to consider system load, capacity, and costs. Likewise, Zukerman et al. [68, 1, 69] as well as others (e.g., [59, 34, 14]) develop decision-theoretic utility models to balance cost and benefit and for use in evaluation.

Elsewhere [23] we have shown evidence that the patterns of activity of a user can change over time. The same is true of Web users. As the user's interests change, the activity recorded may also change; there is no need to revisit a site if the user remembers the content, or if the user now has a different concern. Thus, the users themselves provide a source of change over time in the patterns found in usage logs. However, in additional experiments on these traces [21], we found that there is a second source of change that is even more significant – changes in content. A user's actions (e.g., visiting a page) depend upon the content of those pages. If the pages change content or links, or are removed, then the request stream generated by the user will also change (corresponding to a 2–5% relative cost in accuracy). Thus we conclude that as long as Web access is primarily an information-gathering process, server-side changes will drive changes in user access patterns.

Finally, the astute reader may have noticed that while we mentioned proxy workloads for completeness back in Sect. 4, we have not included them in the figures shown, as they do not provide significant additional information. While running the UCB client workloads and calculating the client-side predictive accuracy (which is what we report), we also calculated the accuracy that a proxy-based predictor would have achieved. In general, its accuracy was 5–10% higher, relative to the client accuracy (i.e., 0.1–2% in absolute terms). A first-order Markov model allowing just one guess, without thresholds, has an asymptotic predictive accuracy of close to 25% for the full UCB proxy trace.

## 8 Alternative Prediction Mechanisms

The history-based mechanisms discussed so far have one significant flaw – they cannot predict a request never made previously. Therefore, it is worth considering alternative mechanisms and combining the predictions from different algorithms.

Our motivating goal is to accurately predict the next request that an individual user is likely to make on the WWW. Therefore, we want to know how helpful Web page content can be in making predictions for what will be requested next. Elsewhere [20, 21] we experimentally validated widespread assumptions that Web pages are typically linked to pages with related textual content, and more specifically, that the anchor text was a reasonable descriptor for the page to which it pointed. Many applications already take advantage of this Web characteristic (e.g., for indexing terms

not on a target page [7] or to extract high-quality descriptions of a page [2, 3]), and here we too exploit this property of the Web.

Relatively few researchers have considered using anything other than simplistic approaches to prediction using Web page content. Given that Web HTML contains links that are followed, we can consider predicting one or more of those links [43, 12, 38, 35, 54]. But which links? One cannot predict all links of a page, since the number of links per page can be quite large. A slightly more intelligent approach is to predict the links of a page, in HTML source order from first to last (corresponding generally to links visible from top to bottom, e.g., [12]). Elsewhere [21, 22] we compare (using a full-content Web log) those simple approaches with an information retrieval-based one that ranks the list of links using a measure of textual similarity to the set of pages recently accessed by the user. In summary, we found that textual similarity-based predictions outperform the simpler approaches: a content-based approach was found to be 29% better than random link selection for prediction, and 40% better than not prefetching in a system with an infinite cache. Finally, since content-based approaches can make predictions even when history-based mechanisms cannot, we found that combining the predictions from both algorithms resulted in increased predictive accuracy (achieving 85–90% of the sum of the individual accuracies).

## 9 Summary

In this chapter we have examined in detail the problem of predicting Web requests, focusing on Markov and Markov-like algorithms that build models from past histories of requests. We have discussed the problems of evaluation, and have provided a consistent description of algorithms used by many researchers in this domain. We used generalized codes that implement the various algorithms described to test the prediction algorithms on the same set of Web traces, facilitating performance comparisons.

We demonstrated the potential for using probabilistic Markov and Markov-like methods for Web sequence prediction. By exploring various parameters, we showed that larger models incorporating multiple contexts could outperform simpler models. We also cited evidence when Web content changes, models learning user behavior should provide some emphasis on recent activity (to adapt to the changes). Finally, we demonstrated the performance changes that result when using Web-inspired algorithm changes. Based on these results, we recommend the use of a multicontext predictor (such as PPM, or the multicontext  $n$ -gram approach described in this chapter). However, for the Web, it appears that relatively few contexts are needed;  $n$ -grams do not need to be much more than 3 or 4. We saw that the utilization of thresholds on prediction can significantly reduce the likelihood of erroneous predictions. Finally, we noted that history-based algorithms are not able to make predictions in all scenarios, and so we described experiments on the utility of content-based predictions.

## 10 Open Questions and Future Work

While this chapter has explored many aspects of learning Web request patterns, many open questions remain for future work. We list a few here:

- What is the trade-off of complexity and storage versus predictive accuracy? Laird and Saul [39, 40] and more recently Chen and Zhang [11] have started to answer this question.
- What is the role of client-side caching on model accuracy? Since a proxy or server-based model does not see requests satisfied by the browser cache, the model they build is not entirely accurate. As we mention at the end of Sect. 2.4, one possibility is to tell the upstream server about requests satisfied by the downstream cache. The important question, though, is how much an effect this extra information will have on the performance of the prediction model.
- What is the effect of using HTTP referrer tags in model generation? Like the previous question, this issue is concerned in part with the client cache. The HTTP referrer tag automatically provides a view into the cache, as it lists the source of the link being requested. This may be a page satisfied by a downstream cache and thus is not part of the request stream seen by the model generator. In particular, it helps capture some uses of the browser back-button and, if used, could build a better model as the current click could be credited to the page on which the link was found, rather than the most recent request in the usage log.
- What is the role that request time stamps can play in model generation and accuracy? Some researchers assume that non-HTML requests received sufficiently near an HTML request (as shown by the time stamps) represent embedded objects and are thus not considered separate clicks (as discussed in Sect. 2.1). However, time stamps might also be used to model links of various types. Two requests very near each other in time are easily believed to be highly associated with each other. Likewise, if two sequential requests are far apart, one might assume that a new session has started, and thus are unrelated. Intermediate values of association might be possible for requests somewhat nearby, possibly even when other requests are found in between.
- What are appropriate evaluation methods (such as those that are tied to real-world utility)? In the end, predictive models likely need to be evaluated within the context of their use. That is, for prefetching, how much improvement in user-perceived response time is realized with various prediction approaches? Similarly, what improvements in user satisfaction (or increased purchases) occur with personalized pages? It is not clear whether the improvement in performance in the end system consistently matches the improvement in predictive performance.
- What is the effect on prediction of classifying user browsing modes? If a user is known to be performing a regular activity (such as reading the daily sports headlines), a user-specific model of behavior might be more appropriate than a global model. Past researchers [9, 33, 13, 32] have attempted to categorize browsing modes (into classes such as surfing, searching, etc.), which may provide assistance in predicting future requests.

## Acknowledgments

This work was supported in part by NSF grant ANI 9903052.

## References

1. D.W. Albrecht, I. Zukerman, and A.E. Nicholson. Pre-sending documents on the WWW: A comparative study. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, volume 2, pages 1274–1279, Stockholm, Sweden, 1999.
2. E. Amitay. Using common hypertext links to identify the best phrasal description of target Web documents. In *Proceedings of the SIGIR'98 Post-Conference Workshop on Hypertext Information Retrieval for the Web*, Melbourne, Australia, 1998.
3. E. Amitay and C. Paris. Automatically summarising Web sites — is there a way around it? In *Proceedings of the Ninth ACM International Conference on Information and Knowledge Management (CIKM 2000)*, Washington, DC, November 2000.
4. T.C. Bell, J.G. Cleary, and I.H. Witten. *Text Compression*. Prentice Hall, Englewood Cliffs, NJ, 1990.
5. A. Bestavros. Using speculation to reduce server load and service time on the WWW. In *Proceedings of the Fourth ACM International Conference on Information and Knowledge Management (CIKM'95)*, Baltimore, MD, November 1995.
6. A. Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time for distributed information systems. In *Proceedings of the International Conference on Data Engineering (ICDE'96)*, New Orleans, LA, March 1996.
7. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the Seventh International World Wide Web Conference*, Brisbane, Australia, April 1998.
8. C. Brodley and R. Kohavi. KDD Cup 2000. Online at <http://www.ecn.purdue.edu/KDDCUP/>, 2000.
9. L.D. Catledge and J.E. Pitkow. Characterizing browsing strategies in the World Wide Web. *Computer Networks and ISDN Systems*, 26(6):1065–1073, 1995.
10. X. Chen and X. Zhang. Coordinated data prefetching by utilizing reference information at both proxy and Web servers. *Performance Evaluation Review*, 29(2):32–38, September 2001. Presented at the 2nd ACM Workshop on Performance and Architecture of Web Servers (PAWS-2001).
11. X. Chen and X. Zhang. A popularity-based prediction model for Web prefetching. *IEEE Computer*, 36:63–70, March 2003.
12. K.-I. Chinen and S. Yamaguchi. An interactive prefetching proxy server for improvement of WWW latency. In *Proceedings of the Seventh Annual Conference of the Internet Society (INET'97)*, Kuala Lumpur, June 1997.
13. C.W. Choo, B. Detlor, and D. Turnbull. Working the Web: An empirical model of Web use. In *33rd Hawaii International Conference on System Science (HICSS)*, Maui, Hawaii, January 2000.
14. E. Cohen, B. Krishnamurthy, and J. Rexford. Efficient algorithms for predicting requests to Web servers. In *Proceedings of IEEE INFOCOM*, New York, March 1999.
15. M.E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.
16. C.R. Cunha. *Trace analysis and its applications to performance enhancements of distributed information systems*. PhD thesis, Computer Science Department, Boston University, 1997.

17. C.R. Cunha, A. Bestavros, and M.E. Crovella. Characteristics of WWW client-based traces. Technical Report TR-95-010, Computer Science Department, Boston University, July 1995.
18. C.R. Cunha and C.F.B. Jaccoud. Determining WWW user's next access and its application to prefetching. In *Proceedings of Second IEEE Symposium on Computers and Communications (ISCC'97)*, Alexandria, Egypt, July 1997.
19. K.M. Curewitz, P. Krishnan, and J.S. Vitter. Practical prefetching via data compression. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, pages 257–266, May 1993.
20. B.D. Davison. Topical locality in the Web. In *Proceedings of the 23rd Annual ACM International Conference on Research and Development in Information Retrieval (SIGIR 2000)*, pages 272–279, Athens, Greece, July 2000.
21. B.D. Davison. *The Design and Evaluation of Web Prefetching and Caching Techniques*. PhD thesis, Department of Computer Science, Rutgers University, New Jersey, October 2002.
22. B.D. Davison. Predicting Web actions from HTML content. In *Proceedings of the Thirteenth ACM Conference on Hypertext and Hypermedia (HT'02)*, pages 159–168, College Park, MD, June 2002.
23. B.D. Davison and H. Hirsh. Predicting sequences of user actions. In *Predicting the Future: AI Approaches to Time-Series Problems*, pages 5–12, Madison, WI, July 1998. AAAI Press. Proceedings of AAAI-98/ICML-98 Workshop, published as Technical Report WS-98-07.
24. B.D. Davison and V. Liberatore. Pushing politely: Improving Web responsiveness one packet at a time (extended abstract). *Performance Evaluation Review*, 28:43–49, September 2000. Presented at the Performance and Architecture of Web Servers (PAWS) Workshop, June 2000.
25. M. Deshpande and G. Karypis. Selective Markov models for predicting Web-page accesses. In *Proceedings of the First SIAM International Conference on Data Mining (SDM'2001)*, Chicago, April 2001.
26. D. Duchamp. Prefetching hyperlinks. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS '99)*, Boulder, CO, October 1999.
27. G. Fan and X.-G. Xia. Maximum likelihood texture analysis and classification using wavelet-domain hidden markov models. In *Proceedings of the 34th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, 2000.
28. L. Fan, Q. Jacobson, P. Cao, and W. Lin. Web prefetching between low-bandwidth clients and proxies: Potential and performance. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '99)*, Atlanta, GA, May 1999.
29. D. Foygel and D. Strelow. Reducing Web latency with hierarchical cache-based prefetching. In *Proceedings of the International Workshop on Scalable Web Services (in conjunction with ICPP'00)*, Toronto, August 2000.
30. S.D. Gribble. UC Berkely Home IP HTTP traces. Online: <http://www.acm.org/sigcomm/ITA/>, July 1997.
31. S.D. Gribble and E.A. Brewer. System design issues for Internet middleware services: Deductions from a large client trace. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS '97)*, December 1997.
32. J. Heer and E.H. Chi. Identification of Web user traffic composition using multi-modal clustering and information scent. In *Proceedings of the Workshop on Web Mining, SIAM Conference on Data Mining*, pages 51–58, Chicago, IL, April 2001.

33. J.H. Hine, C.E. Wills, A. Martel, and J. Sommers. Combining client knowledge and resource dependencies for improved World Wide Web performance. In *Proceedings of the Eighth Annual Conference of the Internet Society (INET'98)*, Geneva, Switzerland, July 1998.
34. E. Horvitz. Continual computation policies for utility-directed prefetching. In *Proceedings of the Seventh ACM Conference on Information and Knowledge Management*, pages 175–184, Bethesda, MD, November 1998. ACM Press, New York.
35. T.I. Ibrahim and C.-Z. Xu. Neural net based pre-fetching to tolerate WWW latency. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS2000)*, April 2000.
36. Z. Jiang and L. Kleinrock. An adaptive network prefetch scheme. *IEEE Journal on Selected Areas in Communications*, 16(3):358–368, April 1998.
37. D. Joseph and D. Grunwald. Prefetching using Markov predictors. *Transactions on Computers*, 48(2), February 1999.
38. R.P. Klemm. WebCompanion: A friendly client-side Web prefetching agent. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):577–594, July/August 1999.
39. P. Laird. Discrete sequence prediction and its applications. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, Menlo Park, CA, 1992. AAAI Press.
40. P. Laird and R. Saul. Discrete sequence prediction and its applications. *Machine Learning*, 15(1):43–68, 1994.
41. B. Lan, S. Bressan, and B.C. Ooi. Making Web servers pushier. In *Proceedings of the Workshop on Web Usage Analysis and User Profiling*, San Diego, CA, August 1999.
42. I.T. Li, Q. Yang, and K. Wang. Classification pruning for Web-request prediction. In *Poster Proceedings of the 10th World Wide Web Conference (WWW10)*, Hong Kong, May 2001.
43. H. Lieberman. Autonomous interface agents. In *Proceedings of the ACM SIGCHI'97 Conference on Human Factors in Computing Systems*, Atlanta, GA, March 1997.
44. T.M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
45. A.E. Nicholson, I. Zukerman, and D.W. Albrecht. A decision-theoretic approach for pre-sending information on the WWW. In *Proceedings of the 5th Pacific Rim International Conference on Artificial Intelligence (PRICAI'98)*, pages 575–586, Singapore, 1998.
46. L. Bottomley of Duke University. EPA-HTTP server logs, 1995. Available from <http://ita.ee.lbl.gov/html/contrib/EPA-HTTP.html>.
47. V.N. Padmanabhan. Improving World Wide Web latency. Technical Report UCB/CSD-95-875, UC Berkeley, May 1995.
48. V.N. Padmanabhan and J.C. Mogul. Using predictive prefetching to improve World Wide Web latency. *Computer Communication Review*, 26:22–36, July 1996. Proceedings of SIGCOMM '96.
49. T. Palpanas and A. Mendelzon. Web prefetching using partial match prediction. In *Proceedings of the Fourth International Web Caching Workshop (WCW99)*, San Diego, CA, March 1999.
50. M. Perkowitz and O. Etzioni. Adaptive Web sites: an AI challenge. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.
51. M. Perkowitz and O. Etzioni. Adaptive Web sites: Automatically synthesizing Web pages. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, WI, July 1998. AAAI Press.
52. M. Perkowitz and O. Etzioni. Adaptive Web sites. *Communications of the ACM*, 43:152–158, August 2000.
53. P.L. Pirolli and J.E. Pitkow. Distributions of surfers' paths through the World Wide Web: Empirical characterization. *World Wide Web*, 2:29–45, 1999.



54. J.E. Pitkow and P.L. Pirolli. Life, death, and lawfulness on the electronic frontier. In *ACM Conference on Human Factors in Computing Systems*, Atlanta, GA, March 1997.
55. J.E. Pitkow and P.L. Pirolli. Mining longest repeated subsequences to predict World Wide Web surfing. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems*, October 1999.
56. J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.
57. R.R. Sarukkai. Link prediction and path analysis using Markov chains. In *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam, May 2000.
58. S. Schechter, M. Krishnan, and M.D. Smith. Using path profiles to predict HTTP requests. *Computer Networks and ISDN Systems*, 30:457–467, 1998. Proceedings of the Seventh International World Wide Web Conference.
59. R. Sen and M.H. Hansen. Predicting a Web user's next request based on log data. *Journal of Computational and Graphical Statistics*, 12(1), 2003.
60. G. Shafer, P.P. Shenoy, and K. Mellouli. Propagating belief functions in qualitative Markov trees. *International Journal of Approximate Reasoning*, 1(4):349–400, 1987.
61. Z. Su, Q. Yang, Y. Lu, and H.-J. Zhang. WhatNext: A prediction system for Web request using n-gram sequence models. In *First International Conference on Web Information Systems and Engineering Conference*, pages 200–207, Hong Kong, June 2000.
62. N. Swaminathan and S.V. Raghavan. Intelligent prefetching in WWW using client behavior characterization. In *Proceedings of the Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2000.
63. L. Tauscher and S. Greenberg. How people revisit Web pages: Empirical findings and implications for the design of history systems. *International Journal of Human Computer Studies*, 47(1):97–138, 1997.
64. I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, 1999. 2nd edn.
65. Y.-H. Wu and A.L.P. Chen. Prediction of Web page accesses by proxy server log. *World Wide Web: Internet and Web Information Systems*, 5:67–88, 2002.
66. Q. Yang, H.H. Zhang, and I.T. Li. Mining Web logs for prediction models in WWW caching and prefetching. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*, San Francisco, August 2001.
67. M.Z. Zhang and Q. Yang. Model-based predictive prefetching. In *Proceedings of the 2nd International Workshop on Management of Information on the Web — Web Data and Text Mining (MIW'01)*, Munich, Germany, September 2001.
68. I. Zukerman and D.W. Albrecht. Predictive statistical models for user modeling. *User Modeling and User-Adapted Interaction*, 11(1/2):5–18, 2001.
69. I. Zukerman, D.W. Albrecht, and A.E. Nicholson. Predicting users' requests on the WWW. In *Proceedings of the Seventh International Conference on User Modeling (UM-99)*, pages 275–284, Banff, Canada, June 1999.

---

# Index

- REBECA, 339
- 2g-precedence model, 323
- accuracy interval, 324
- active capability, 301
- active databases, 223, 224, 226, 249, 279, 304
  - object-oriented, 227, 304
  - relational, 225, 227
- Active XML, 275
- adaptation model, 358, 391
- adaptation techniques, 360
- adaptive content selection, 360
- adaptive courseware, 402
- adaptive crawlers, 166
- adaptive educational hypermedia, 387
  - architecture, 389
- adaptive hypermedia, 357, 358
  - adaptation engines, 361
  - architecture, 397
  - architectures, 358, 361, 364
  - authoring, 361
  - collaborative authoring, 405
  - goals and constraints model, 405
  - in agent-based environments, 364
  - in e-commerce, 364
  - in education, 364
  - in information retrieval, 364
  - in medicine, 364
  - in tourism, 364
  - in wireless environments, 364
  - information fragments, 373
  - models, 362, 373
  - monitoring rools, 361
  - presentation descriptions, 373, 376
  - profilers, 361
  - requirements, 358
  - systems, 357, 358
- Adaptive Hypermedia Application Model (AHAM), 362
  - adaptive navigation, 360
  - adaptive navigation support, 392
  - adaptive presentation, 360
  - adaptive Web Services, 384
  - agent-based architecture, 305
- AHA, 394
- AHAM, 389
- analysis
  - link, 179
  - of ECA rules, 223, 227, 229
  - of Web communities, 58, 60, 61
  - of XML ECA rules, 228, 239–241
- assessment, 395
- authoring of hypermedia, 361, 387
- authoring support, 402
- automata, 117, 121, 122
- Barabási-Albert model, 70, 81
- Best Trail algorithm, 117, 120, 125, 133, 135–140, 147
- bibliographic metrics, 48
- bipartite cores, 49
- broker networks, self-stabilization, 331
- browser
  - back button, 130, 141
  - history list, 130
  - toolbar, 131
- Bush, Vannevar, 119, 124

- caching of Web pages, 314
  - caching of Web pages, directory-based, 315
  - caching of Web pages, hash-based, 314
  - canonical URL, 159
  - capture-recapture, 30
  - change detection, 301, 307, 310, 311
  - change detection graph, 310
  - change notification, 308
  - change presentation, 316
  - change propagation, 310
  - change specification language, 307
  - click-distance, 417
  - click-distance reduction, 424
  - collapsed contingency table, 36
  - community algorithm, 57
  - comparative explanation, 393
  - composite change, 310
  - concept editor, 400
  - concept-based addressing, 333
  - concept-based publish/subscribe, 332
  - conditional inclusion of fragments, 393
  - content time, 425
  - content-based routing, 330
  - context
    - focused crawler, 165
  - CORA, 171
  - cosine weights, 163
  - coupling modes, 326
  - crawler
    - adaptive, 166
    - best first, 163
    - focused, 153, 164
    - topical, 153
  - crawler performance
    - harvest rate, 169
    - precision, 169
    - recall, 169
  - crawling
    - adaptive, 166
    - algorithms, 163
    - applications, 171
    - as graph search, 155
    - ethics, 158
    - evaluation, 168
    - infrastructure, 155
    - multithreading, 161
    - the Web, 153
  - data integration, 275, 278, 284, 319
    - using Web services, 275
  - data mining, 436
  - Dexter Model, 362
  - directed surfer, 186
  - domain model, 358, 391
  - dynamic
    - bookmarks, 145
    - trails, 145
  - dynamic changes
    - search engines, 195
    - Web, 195
  - ECA rule generation, 309
  - ECA rules, 223, 249, 301, 325, 335, 394
    - execution model, 226, 234, 252, 257, 266
    - for XML, 223, 229, 230, 249, 253, 256
  - Engelbart, Douglas, 125
  - EPA HTTP server log, 447, 448
  - evaluating crawlers, 168
  - event aggregation, 323, 334
  - event algebra, 224, 307, 334
  - event brokers, 330
  - event composition, 323, 334
  - event detection, 224, 251, 291
  - event dissemination, 319, 322
  - event integration, 324
  - event languages, 224, 227, 229, 230, 251, 307
  - event notification, 319
  - event ordering, 323
  - event representation, 332
  - event routing strategies, 330
  - event visibility, 327, 338
  - event, partial order, 334
  - event-condition-action rules, 223, 249, 394
  - event-based application management, 338
  - event-based applications, 319
  - event-based interaction, 328
  - event-based management, 327
  - execution model
    - of ECA rules, 252, 257, 266
  - execution model, of ECA rules, 226, 234
  - filter placement, 319
  - filter-based routing, 330
  - fish search, 125
  - fisheye
    - inverse, 143
    - view, 128, 129

- focused crawler, 153, 164
  - context, 165
- frontier, synchronization, 161
- gain rank, 133
- graph author tool, 401
- guided tour, 123, 129
- HFA, 121, 122
- hierarchical menu system (HMS), 417
- HITS, 52, 103, 179, 181
  - speed of, 182
  - topic drift, 181
  - Web communities, 54
- HMS, 417
- HPA, 122, 142, 143
- HTML, 23, 301
  - tag tree, 160
- HTML parsing, 159
- HTTP, 437, 438, 455
- Hypermedia Design Model (HDM), 362
- Hypertext Finite Automaton (HFA), 121, 122
- Hypertext Probabilistic Automaton (HPA), 122, 142, 143
- indexable Web, 23, 25, 40
- information scent, 126
- InfoSpiders, 127, 171
- integration of notifications and transactions, 326, 336
- intelligent agents, 301
- Intelligent Multimedia Presentations Model (IMMPS), 362
- interaction patterns, 320
- Internet, 69, 73
- Internet-enabled car, 342
- Java Message Service (JMS), 322
- knowledge of concepts, 390
- landmark nodes, 129
- learning objects, 396
- learning objects metadata, 396
- learning standards, 396
- Letizia, 174
- link adaptation, 392
- link analysis, 179
  - for quality, 180
  - for topical relevance, 180
- machine learning, 438
  - batch evaluation, 438
  - online evaluation, 438
  - test set, 438
  - training set, 438
- maps, 128, 129
- Mapuccino, 174
- Markov assumption, 441
- Markov chain, 117, 123, 142–144
  - Monte Carlo, 33
- Markov model, 441, 443, 452, 453
  - $k$ th order, 443
  - $n$ -gram, 441, 448–450, 452
  - first-order, 443, 448, 452, 453
  - path profile, 443
  - point profile, 443
  - second-order, 443
  - zeroth-order, 441
- Markov tree, 441, 448
- maximum flow, 55, 56
- Memex, 119, 124, 125
- meta-auctions, 341
- metrics
  - bibliographic, 48
  - page change, 97
- middleware
  - for event-based applications, 319
  - mediated transactions, 326, 336
- minimum cut, 55, 56
- mobile Internet, 411, 412
- mobile networks
  - 1G, 413
  - 2.5G, 413
  - 2G, 413
  - 3G, 413
  - GPRS, 413
- mobile portal, 411
- mobile portal usability, 417
- mobile Web, 144
- model
  - Rasch, 37
- models of Web usage, 435
  - collaborative filtering, 447
  - user clustering, 447
- Molloy–Reed criterion, 83
- multiple list population estimation, 23
- multiple-choice tests, 395
- Munich model, 389

Music Machines server log, 448  
 MySpiders, 166, 171

navigation, 117

- bar, 129
- engine, 120, 140
- history, 129, 130
- potential, 117
- problem, 117, 119, 120, 125, 145
- sessions, 120, 124, 142
- tools, 128
- tree, 135–138
- tree window, 140
- types of, 118

navigation time, 425

Nelson, Ted, 119, 125

network

- adjacency matrix, 71
- attacks, 83
- characteristics, 71
- clustering coefficient, 71
- connectivity, 71
- degree, 71
- failures, 83
- giant component, 72
- growing, 72
- growing exponential, 79
- percolation on, 83
- resilience, 85
- scale-free, 70, 80
- shortest path, 71
- small-world, 70, 71, 77
- social, 145
- strongly connected component, 72, 75
- weakly connected component, 72, 75

nondeterminism, of Web service calls, 288

notification service, 322, 330

- channel-based, 322
- content-based, 319, 322
- subject-based, 322

Object-Oriented Hypermedia Design Model  
 (OOHDM), 362

ontological structures, 402

ontologies, 324, 333, 359, 402

ontology of course authoring, 402

open distributed systems, 319

overlay model, 364, 390, 391

page change models, 94

page importance, 169

classifier score, 169

relevance, 169

similarity to seed pages, 169

PageRank, 53, 103, 179, 183

query-dependent, 179, 186

topic drift, 184

Web communities, 55

peer-to-peer

architectures, 246, 275

computation, 279, 287

data integration, 275

peer capabilities and security, 289

periodic event, 313

personalized learning experience, 387

personalized menu, 421

personalized navigation, 411, 419

Petersen estimate, 30

Poisson distribution, 70

popularity is attractive concept, 81

portal personalization, 411

portal usability, 411

posterior distribution of the number of Web

pages, 33, 36

potential gain, 117, 120, 131–133, 140, 147

PPM, 444, 450, 452

prediction by partial matching, 444, 450

prediction of Web request patterns, 435

accuracy, 440, 449, 451, 455

confidence, 439, 448, 452

content-based, 454

history-based, 436, 453

macroaverage, 438

microaverage, 438

mistake costs, 451

performance evaluation, 437, 439

precision, 440

prediction window, 445, 451

prediction workloads, 437, 445

support, 439, 448, 452

thresholds, 439, 448

top-*n* predictions, 440

preferential attachment, 81

prefetching of Web pages, 438, 453

preloading of Web pages, 435

prerequisite explanation, 393

primitive change, 310

proxy, 446

- proxy cache, 446
- publicly indexable Web, 23
- publish/subscribe notification service, 319
- pull paradigm, 303
- push technology, 301
- push-based architecture, 304
- push/pull paradigms, 301, 303
- query-dependent PageRank, 179, 186
- random graph
  - classical, 70, 76
  - Erdős-Rényi, 70, 76
- random surfer, 183
- ranking
  - authorities and hubs, HITS, 103
  - link-based, 103
  - PageRank, 53, 103, 179, 183
  - query-dependent PageRank, 179
  - time-biased, 105
- Rasch model, 23, 33, 37, 40
  - Bayesian, 37
  - Bayesian approach, 33, 35
  - generalized linear, 34
- reactive functionality, 223, 319, 325, 335
- regression-based projections, 36
- reliable notification service, 319
- request-response, 389
- resource model, 402
- robot exclusion protocol, 158
- sampling Web pages, 23
- scalability
  - of query-dependent PageRank, 187
- scaling relations, cutoff, 82
- scent, information, 126
- scopes, 319, 327, 338
- search
  - fish, 125
  - shark, 125, 127, 163
  - Web, 179
- search engine, 23, 40
  - architecture, 195
  - evaluation, 195
- search engine coverage, 25
- search engine query, 25
- search performance metrics in dynamic environments, 195
- semantic metadata, 324
- Semantic Web, 324, 359, 384, 402
- semantics, of ECA rules, 224, 226, 234, 252, 257
- sentinel, activation/deactivation, 309
- sequence prediction, 436
- shark search, 125, 127, 163
- size of the World Wide Web, 23, 30
- social networks, 145
- spectral methods, 51
- SQL3, 227, 257
- SSDC server log, 448
- stemming, 160
- stoplisting, 160
- stopwords, 160
- subscription/notification mechanism, 310
- suffix tree, 144
- task ontology, 403
- TDAG, Transition Directed Acyclic Graph, 448
- termination
  - of ECA rules, 227
  - of Web service calls, 285, 288
- TFIDF, 179
- time granularity, 334
- timestamping, 334
- topic drift, 179
  - HITS, 181
  - PageRank, 184
- topical crawlers, 153
- trail records, 142
- trails, 117, 119–126, 129–140, 142–144, 147
  - authored, 123
  - derived, 123
  - dynamic authored, 145
  - emergent, 123, 142
  - evaluation of, 147
  - personal, 123
  - scoring functions for, 133–135
- transactional publish/subscribe, 319, 327, 336
- UCB Home IP HTTP Trace, 446
- update languages, 229–231, 251
- URL canonicalization, 159
- user model, 358, 389, 390
- user modeling, 359
- user profiles, 359
- user request sessions, 438

utility model, 453

Walden's paths, 129

WAP, 414

WAP handset, 414

WAP portal, 411

Watts-Strogatz model, 77

Web applications, 223, 249, 270, 275, 296,  
301, 304, 319, 321, 357, 358, 360, 364,  
384, 387, 411

heterogeneity, 324

Web client, 446

history, 446

Web communities, 45–49, 51–55, 57, 58,  
60–62, 65–67

clustering, 46

flow-based, 55

HITS, 54

PageRank, 55

Web crawl

frontier, 155

history, 157

Web crawling, 153, 189

Web graph, 117, 120, 121, 123, 129, 131,  
133, 142

Web log co-occurrences, 437

embedded object, 437

embedding relationship, 437

referring page, 437

traversal relationship, 437

Web logs, 437

click-stream analysis, 437

HTTP referrer header, 437, 447

Web page repository, 157

Web pages

fetching, 158

Web pages as a closed population, 30

Web prediction, 435, 438

Web proxy, 446

Web request patterns, 435

machine learning, 436

prediction, 435

Web search, 179

Web server, 447

Web Services, 384

Web services, 250, 271, 275, 278  
composition and integration, 278  
continuous, 276, 285  
definition, 284

Web structure, 74, 93, 99

dynamics, 100

site migration, 103

Web view, 142

Web, estimating rate of page changes, 96

Web, freshness, 93, 95, 102

Web, growth models, 99

Web, page quality, 93, 103

Web, recency, 93, 100

Web-based adaptive systems, 389

Web-Watcher, 126, 127

WebML, 362

WebVigiL, 301

architecture, 305

wrapper-based architecture, 305

X<sup>2</sup>TS, 340

XAHM, 370

abstract Concepts, 373

adaption space, 371

adaptivity dimensions, 371

architecture, 382

author module, 383

elementary abstract concepts, 373

metadata, 374

metadescriptions, 374

modeling phases, 378

probabilistic interpretation of hypermedia,  
377

run-time system, 382

user classification, 378

Xanadu, 119, 125

XML, 223, 249, 275, 277, 301

ECA rules for, 223, 229, 249, 253, 256

XML Adaptive Hypermedia Model  
(XAHM), 370

XQuery ECA rules, 249, 253, 256